



Unternehmen für Bildung.

Daten systemübergreifend
bereitstellen

Modul: **ITG / I**

Version: 1.0
Oktober 2022

Erstellt von: IT-Abteilung

Modul: **ITG / I - Daten systemübergreifend bereitstellen**

Version, Datum: 1.0, Oktober 2022

Erstellt von: IT-Abteilung

Alle Rechte vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne Genehmigung des Herausgebers reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Die Modulunterlage und alle ihre Bestandteile wurden mit großer Sorgfalt erstellt und geprüft. Dennoch können Fehler nicht vollkommen ausgeschlossen werden. Der Herausgeber und die Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Literaturhinweis

Titel: IT-Berufe Grundstufe

ISBN: 978-3-14-220000-2

Verlag: Westermann Gruppe

Inhalt

TAG 1 - Kundenaufträge im Rahmen von Softwareprojekten bearbeiten.....	7
LE 1 - Das Basiswissen über Projekte und Projektmanagement wiederholen und festigen.....	7
LE 2 - Softwareprojekte mithilfe von agilen Vorgehensmodellen umsetzen	8
LE 3 - Scrum	10
LE 4 - Kanban	16
LE 5 - Extreme Programming (XP)	18
TAG 2 - Daten bewerten und aus verschiedenen Datenquellen zusammenführen.....	20
LE 1 - Die Qualität von Daten beschreiben	20
TAG 3 - Daten aus Datenquellen abrufen	25
LE 1 - Datenformate zur Datenübertragung	26
LE 2 – Datenzugriff und -Übertragung	29
LE 3 - Daten heterogener Datenquellen zusammenführen	31
TAG 4 - Objektorientierte Softwarelösungen unter Beachtung der Informationssicherheit planen	34
LE 1 - Programmierparadigmen unterscheiden.....	34
LE 2 - Imperative Programmierparadigmen.....	35
LE 3 - Deklarative Programmierparadigmen	36
TAG 5 - Das objektorientierte Programmierparadigma beschreiben	38
LE 1 - Abstraktion	38
LE 2 - Kapselung	39
LE 3 - Vererbung.....	40
LE 4 - Polymorphie	40
LE 5 - Objektorientierte Programmiersprachen	41
TAG 6 - Die Modellierungssprache UML verwenden	43
LE 1 - Anwendungsfalldiagramme beschreiben und anwenden	47
LE 2 - Umsetzung eines Anwendungsfalldiagramms für die Ausgangssituation	48
TAG 7 - Klassendiagramme beschreiben und anwenden.....	50
LE 1 - Umsetzung eines Klassendiagramms für die Ausgangssituation	53
LE 2 - Weitere Beispielaufgabe zum Klassendiagramm	53
TAG 8 - Aktivitätsdiagramm beschreiben und anwenden	55
LE 1 - Notationselemente und Aufbau	55
LE 2 - Umsetzung eines Aktivitätsdiagramms für die Ausgangssituation	57
TAG 9 - Sicherheitsrelevante Aspekte bei der Softwareplanung berücksichtigen	59
LE 1 - Benutzerschnittstellen unter softwareergonomischen Gesichtspunkten planen.....	63
LE 2 - Grundlegende Anforderungen an Benutzerschnittstellen präsentieren	64
LE 3 - Eine Oberfläche für eine Benutzerschnittstelle entwerfen	66
TAG 10 - Vertiefung des Wissens und Modulprüfung Teil 1	69
LE 1 - Einleitung.....	69
LE 2 – Modulprüfung, Teil 1	69
TAG 11 - TAG 13 - Anwendungen in Java implementieren	70
LE 1 - Java beschreiben und eine Entwicklungsumgebung auswählen.....	70
LE 2 - Grundlegende Sprachelemente beschreiben und Konsolenanwendungen implementieren ..	72
LE 3 - Grundlegende Sprachelemente	73

LE 4 - Ein- und Ausgabe in der Konsole.....	77
LE 5 - Kontrollstrukturen	78
LE 6 - Funktionen.....	79
LE 7 - Felder und Zufallszahlen	80
LE 8 - Exception Handling	81
TAG 14 – TAG 16 - Das objektorientierte Programmierparadigma in Java umsetzen	84
LE 1 - Klassen, Eigenschaften, Methoden, Zugriffsmodifizier und Objekte	84
LE 2 - Vererbung.....	87
LE 3 - Überladen und Überschreiben von Methoden	88
LE 4 - Abstrakte Klasse und Interface	89
LE 5 - Weitere Beispielaufgabe	92
TAG 17 & TAG 18 - Grafische Benutzerschnittstellen in Java entwickeln	94
LE 1 - Grundlagen.....	94
LE 2 - Nutzung des WindowBuilder	96
TAG 19 & TAG 20 - Anwendungen in Python implementieren	103
LE 1 - Das objektorientierte Programmierparadigma in Python umsetzen	103
LE 2 - Klassen, Eigenschaften, Methoden, Zugriffsmodifizier und Objekte	103
LE 3 - Vererbung.....	107
LE 4 - Überladen und Überschreiben von Methoden	108
LE 5 - Abstrakte Klasse und Interface	109
LE 6 - Weitere Beispielaufgabe	110
TAG 21 - Grafische Benutzerschnittstellen in Python entwickeln	112
LE 1 - Grundlagen.....	112
LE 2 - Weitere Beispielaufgabe	113
TAG 22 - Datenbanklösungen bedarfsgerecht entwickeln	117
LE 1 - Ein geeignetes Datenmodell auswählen	117
LE 2 - Den Prozess des Designs relationaler Datenbanken beschreiben.....	121
TAG 23 - Relationale Datenmodelle normalisieren	124
LE 1 - Erste Normalform (1. NF).....	124
LE 2 - Zweite Normalform (2. NF).....	125
LE 3 - Dritte Normalform (3. NF).....	125
LE 4 - Weitere Normalformen und Nachteile der Normalisierung	126
LE 5 - Weitere Beispielaufgabe	127
TAG 24 - Das Basiswissen über SQL erweitern und anwenden.....	128
LE 1 - Abfragen über mehrere Tabellen	128
LE 2 - Kombinieren von Abfragemengen.....	130
LE 3 - Unterabfragen	130
LE 4 - Rechtevergabe	131
TAG 25 - Eine MySQL-Datenbank mit Java ansprechen.....	133
TAG 26 - NoSQL-Datenbanken und deren Datenmodelle unterscheiden	136
LE 1 - Dokumentenorientierte Datenbank	136
LE 2 - Graphdatenbank.....	137
LE 3 - Key-Value-Datenbank	137

LE 4 - Spaltenorientierte Datenbank.....	138
TAG 27 - Die NoSQL-Datenbank „MongoDB“ mit Python ansprechen.....	139
TAG 28 - Cloud-basierte Datenbanklösungen unterscheiden.....	140
TAG 29 - Software testen und dokumentieren	143
LE 1 - Blackbox-Tests planen	143
LE 2 - Testdatengeneratoren verwenden	145
LE 3 - Projektabnahmen durchführen und Abnahmeprotokolle erstellen	148
TAG 30 - Vertiefung des Wissens und Modulprüfung Teil 2	150
LE 1 - Einleitung.....	150
LE 2 - Modulprüfung	150

In der Zeit von Big Data und exponentiellen Datenwachstums ist es wichtig, die anfallenden Daten entsprechend zu verarbeiten, zu speichern und anschließend für das Unternehmen sinnvoll zu nutzen. Dazu müssen die verfügbaren Daten und die dazugehörigen Quellen zunächst einmal analysiert und hinsichtlich ihrer Qualität und Verfügbarkeit bewertet werden. Im Anschluss daran werden Softwarelösungen entwickelt, welche diese Daten aus den verschiedenen Quellen zusammenführen und für eine weitere Analyse systemübergreifend bereitstellen. Bei diesem Prozess verlassen sich viele Unternehmen noch auf klassische Konzepte, welche auch in vielen Bereichen noch ihre Berechtigung haben. Dies trifft vor allem dort zu, bei denen moderat wachsende, strukturierte Datenmengen verarbeitet und gespeichert werden müssen. Diese Konzepte stoßen aber in Bereichen, in denen große, unstrukturierte Datenmengen anfallen, schnell an ihre Grenzen und erweisen sich als zeit- und ressourcenraubend. Deswegen beginnen immer mehr Unternehmen, die klassischen Wege zu verlassen und setzen sowohl bei der Softwareentwicklung als auch bei der Datenintegration auf moderne und agile Ansätze wie Cloudbasierte NoSQL- Datenbanklösungen oder agile Vorgehensmodelle, z. B. Scrum. Um die Qualität der zu entwickelnden Software zu gewährleisten, wird heutzutage zunehmend auf das objektorientierte Programmierparadigma gesetzt und es kommen Programmiersprachen zum Einsatz, welche dieses Paradigma unterstützen, z. B. Java oder Python. In diesem Zusammenhang spielt auch die moderne Modellierungssprache UML zur Planung entsprechender Softwarelösungen eine große Rolle. Auch eine sorgfältige Planung und ergonomische Gestaltung der Benutzerschnittstellen gewinnt zunehmend an Bedeutung. Alle diese Themen werden in diesem Lernfeld angesprochen und näher betrachtet. Dabei orientieren sich die Inhalte vom Schüler- und vom Arbeitsbuch an der Ausbildungsverordnung und am Rahmenlehrplan der neuen IT- Berufe.

TAG 1 - Kundenaufträge im Rahmen von Softwareprojekten bearbeiten

Die Bearbeitung von Kundenaufträgen im Rahmen von Software Projekten gewinnt immer mehr an Bedeutung. Der Beitrag der Projektarbeit zum Bruttoinlandsprodukt lag im Jahre 2007 noch bei ca. 2% des Bruttoinlandsproduktes. 2020 lag dieser Wert schon bei 15% und die Tendenz zeigt weiter nach oben. Daran ist zu erkennen, wie wichtig die Arbeit in Projekten für Unternehmen geworden ist. In diesem Zusammenhang ist es erforderlich, dass diese ein geeignetes Projektmanagement entwickeln und pflegen, um den zunehmend komplexen Anforderungen der Kunden gerecht zu werden.

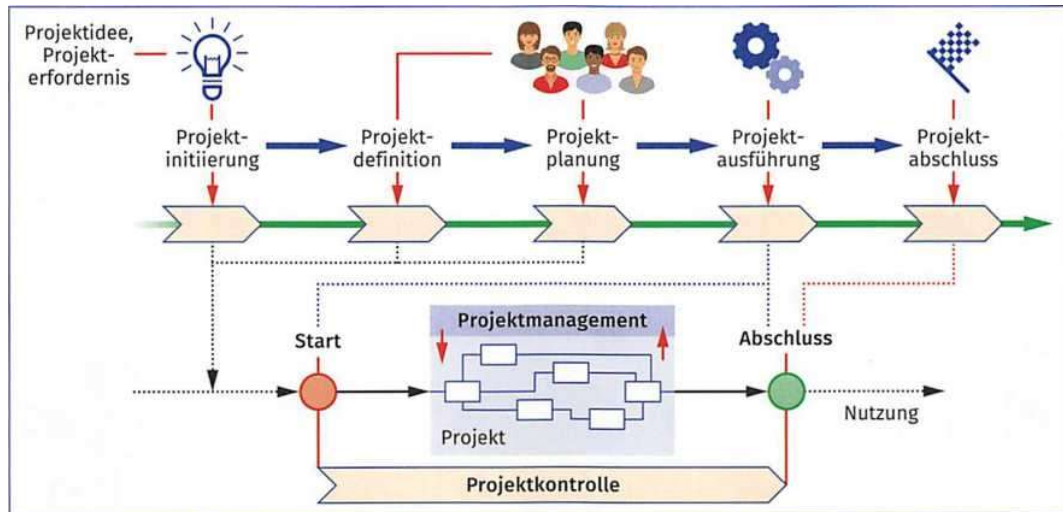


LE 1 - Das Basiswissen über Projekte und Projektmanagement wiederholen und festigen

Die Arbeit in Projekten unterscheidet sich wesentlich von den alltäglichen Routinearbeiten. Vergleicht man die Definition des Begriffs „Projekt“ für verschiedene Projektmanagementstandards, so lässt sich als Gemeinsamkeit festhalten: Projekte sind Vorhaben, die zeitlich begrenzt sind, ein klar umrissenes Zielbild und Einmaligkeit aufweisen.



Das Thema der Projektaufgabe (z. B. Datenbankentwicklung, Webentwicklung, Erstellung einer App usw.) oder die Komplexität spielt bei der Projektdefinition keine Rolle. Bei Software Projekten kann es sich beispielsweise um Neu- oder Anpassungsentwicklungen handeln, welche für das Unternehmen selbst erstellt werden oder von einem externen Auftraggeber kommen.



Projekte durchlaufen i.d.R. verschiedene Phasen. Die Projektphasen stellen in sich abgeschlossene Zeiträume innerhalb des Projektes dar und können unterschiedlich gestaltet sein.

Eine entscheidende Rolle bei der Durchführung von Projekten spielt das Projektmanagement. Unter Projektmanagement ist die Planung, die Überwachung, die Steuerung und der Abschluss eines Projektes zu verstehen. Jedes Projekt ist dabei ein Balanceakt zwischen Umfang, Zeit und Kosten. Die Änderung eines dieser Faktoren zieht unweigerlich eine Änderung der anderen Faktoren nach sich. Das Projektmanagement als Methode entstammt dem Ingenieurwesen und wurde von Beginn an fachübergreifend konzipiert und weiterentwickelt.



LE 2 - Softwareprojekte mithilfe von agilen Vorgehensmodellen umsetzen

Ein erfolgreiches Softwareprojekt zeichnet sich dadurch aus, dass es im geplanten Zeitrahmen, innerhalb der definierten Kosten und in der gewünschten Qualität fertiggestellt wird. Um dieses zu gewährleisten, wurden verschiedene Vorgehensmodell entwickelt, welche sich in klassische (Wasserfallmodell, V-Modell) und in agile Modelle (Scrum, DevOps-Ansatz) einteilen lassen. Die klassischen Vorgehensmodelle sind vor allem durch eine Abfolge von Projektphasen gekennzeichnet, welche durch Meilensteine getrennt sind. Nachdem die Kosten, die Termine und der Personaleinsatz geplant sind, wird versucht, Änderungen im Projektablauf so weit wie möglich zu vermeiden, da diese i.d.R. sehr aufwendig sind.



Dieser Ansatz der klassischen Vorgehensmodelle wird sehr oft als schwerfällig und bürokratisch wahrgenommen. In vielen Fällen wird am wirklichen Bedarf des Kunden vorbeigeplant. Die Schwierigkeiten, welche bei den klassischen Vorgehensmodellen auftreten, führten in der Vergangenheit immer wieder dazu, dass Softwareprojekte begonnen, aber nicht abgeschlossen

wurden. Deswegen rückten mit der Zeit die agilen Vorgehensmodelle immer mehr in den Fokus der Softwareentwicklung.

Agile Vorgehensmodelle versprechen schnelle flexible Projekte, die mit hoher Qualität besser den Bedürfnissen der Kunden entsprechen. Dazu wird versucht, die Entwurfsphase auf ein Mindestmaß zu reduzieren und im Entwicklungsprozess so früh wie möglich zu ausführbarer Software zu gelangen. Dabei wird möglichst von Anfang an der Nutzer bzw. der Auftraggeber in den Entwicklungsprozess mit einbezogen, sodass man schnell und flexibel auf Kundenwünsche eingehen kann und die Kundenzufriedenheit insgesamt erhöht wird.

Leitlinien des „Agilen Manifests“ (Manifesto for Agile Software Development)

Das „Agile Manifest“ wurde im Jahr 2001 im Bereich der Softwareentwicklung ausgearbeitet und formuliert Verhaltensregeln und Werte für agile Teams. Die vier Leitsätze besagen Folgendes: „Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen. Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:

Individuen und Interaktionen	sind wichtiger als	Prozesse und Werkzeuge.
Funktionierende Software	ist wichtiger als	umfassende Dokumentation.
Zusammenarbeit mit dem Kunden	ist wichtiger als	Vertragsverhandlungen.
Reaktion auf Veränderung	ist wichtiger als	das Befolgen eines Plans.

Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein."

Neben den Vorteilen, welche unzweifelhaft die agilen Vorgehensmodelle in der Softwareentwicklung liefern, gibt es auch Nachteile zu erwähnen. Zum einen beanspruchen die verschiedensten Stand-up-Meetings eine nicht unerhebliche Menge an Zeit, welche als Programmierzeit verloren geht. Weiterhin wird sehr viel getestet, was zu nicht zu unterschätzenden Kosten führen kann. Dabei sind die Projektkosten ohnehin schon sehr schwer zu kalkulieren. Auch der ständige Kundenkontakt gestaltet sich in einigen Fällen schwierig und ist nicht immer in der Form gewünscht, wie es die agile Vorgehensweise erforderlich macht.

Vergleich der Umsetzung wichtiger Punkte bei klassischen und agilen Vorgehensmodellen	
Klassisch	Agil
Anforderungen zu Beginn bekannt	Anforderungen zu Beginn unscharf
Änderungen von Anforderungen während Projektverlauf schwierig	Änderungen an Anforderungen während Projektverlauf eingeplant
Hohe Kosten für späte Anforderungsänderungen	Geringe Kosten für späte Anforderungsänderungen
Sequenzieller Entwicklungsprozess	Iterativer Entwicklungsprozess
Kunde sieht nur Endergebnis	Kunde erhält Zwischenergebnisse
Große Teams möglich	Relativ kleine Teams nötig
Klare Hierarchie	Selbstorganisierte Teams
Viel Kommunikation über Dokumente und lange Meetings	Viel informelle Kommunikation und Stand-up-Meetings
Aufwandsschätzung durch Projektleiter oder Experten	Aufwandsschätzung gemeinsam im Team

Bei Erhebungen über den Einsatz verschiedener Vorgehensmodelle bei deutschen Unternehmen zeigt sich ein klarer Trend hin zu den agilen Vorgehensmodellen. Das mit Abstand am

meisten eingesetzte Vorgehensmodell ist Scrum, weswegen es hier als erstes einer näheren Betrachtung unterzogen wird.

LE 3 - Scrum

Scrum gehört zu den agilen Vorgehensmodellen im Rahmen des Projektmanagements. Ursprünglich für die Softwareentwicklung konzipiert, wird Scrum seit vielen Jahren in den unterschiedlichsten Bereichen der Produktentwicklung verwendet und hat sich dabei zu dem am häufigsten eingesetzten Vorgehensmodell entwickelt. Obwohl die Vorgehensweise in Scrum relativ einfach zu verstehen ist, erweist sich eine erfolgreiche Umsetzung in der Praxis als eine nicht zu unterschätzende Herausforderung.

In Scrum gibt es drei grundlegende Rollen: den Product Owner, den Scrum Master und den Entwickler als Mitglied des Entwicklungsteams.



Rollen bei Scrum

Product Owner

Der Product Owner ist eine Person, die alle Anforderungen an das Produkt formuliert, darunter das übergeordnete Ziel, das auch Vision genannt wird. Auch die Stakeholder können ihre Anforderungen an den Product Owner herantragen. Die Entscheidung, welche Anforderungen umgesetzt werden, trifft der Product Owner aber allein. Der Product Owner kann der Kunde selbst oder dessen Vertreter sein. Er ist dafür verantwortlich, dass alle Anforderungen an das Produkt eindeutig formuliert und priorisiert werden, sodass sie vom Entwicklungsteam verstanden werden. Dies geschieht durch das sogenannte Product Backlog, für dessen Pflege der Product Owner ebenfalls zuständig ist.

Scrum Master

Der Scrum Master ist Teil des Scrum-Teams und seine Aufgabe besteht darin, dafür zu sorgen, dass das Entwicklungsteam problemlos arbeiten kann. Dazu organisiert er die nötigen Ressourcen, ist Moderator und dient als Ansprechpartner. Weiterhin gehört zu den Aufgaben des Scrum Masters die Sicherstellung der von Scrum einzuhaltenden Regeln, Praktiken und Prozesse. Dazu schult er u.a. die Teammitglieder und unterstützt die Entwicklung eines Bewusstseins für die dynamische Arbeitsweise mit Scrum.

Entwicklerteam

Das Entwicklerteam besteht aus drei bis neun qualifizierten Fachkräften, von denen jedes Mitglied mehrere Aufgaben im Projekt übernehmen kann. Die Teammitglieder haben selten ihre Spezialgebiete, sondern es wird angestrebt, dass sich jeder durch alle Bereiche bewegen kann. Das Team organisiert seine Aufgaben selbst. Das Entwicklungsteam ist gemeinsam für die gelieferten Ergebnisse verantwortlich.

Für die Arbeit in Scrum-Projekten wurden drei wesentliche Grundsätze entwickelt, um die Qualität des Endproduktes zu gewährleisten und das Projekt erfolgreich abzuschließen.

Grundsätze bei Scrum

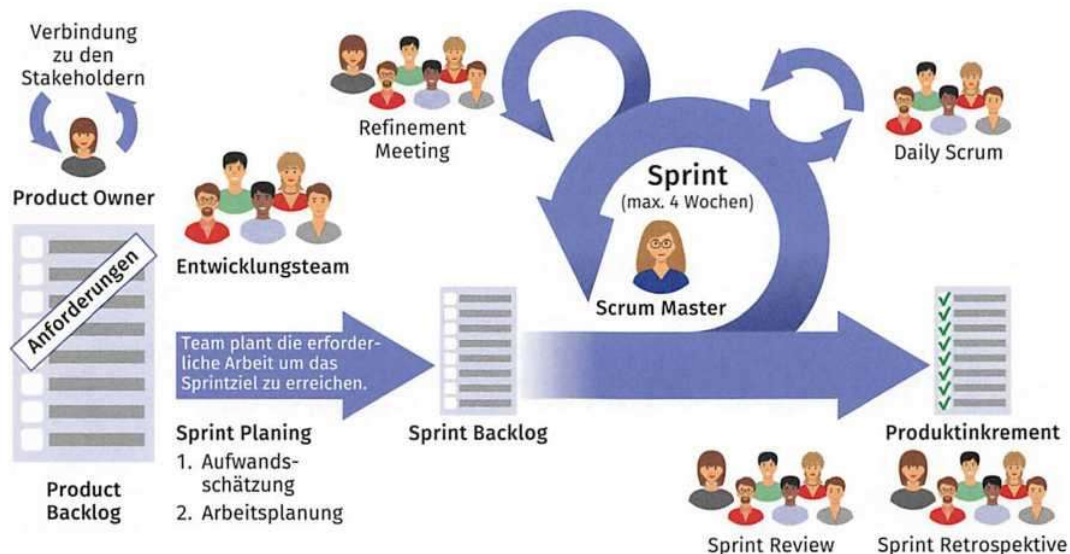
Transparency (Transparenz): Alle entwicklungsrelevanten Informationen müssen für alle bekannt gemacht und festgehalten werden. Dieses betrifft z. B. Projektfortschritte, auftretende Probleme oder Design-Richtlinien.

Inspection (Überprüfung): In regelmäßigen Abständen werden die Scrum-Artefakte, die Projektfortschritte und die Zusammenarbeit im Team bewertet und Feedback gegeben.

Adaption (Anpassung): Auf Grundlage der bei der Überprüfung gewonnenen Erkenntnisse werden Entwicklungspläne und Vorgehensweisen an die neuen Bedingungen angepasst.

Scrum-Prozess

Scrum läuft in sogenannten Sprints ab. Sprints sind feste Intervalle, an deren Ende ein Zwischenergebnis geliefert werden soll. Bevor diese durchgeführt werden können, müssen das Projekt grob geplant und entsprechende Vorbereitungen getroffen werden, damit das gesamte Projektteam ein gemeinsames Verständnis für die Ziele und Meilensteine des Projektes hat.



Am Anfang des Scrum-Prozesses steht eine Idee, welche umgesetzt werden soll. Am Ende des Prozesses gibt es Zeit für Reflexionen und die Erkenntnis, dass ein funktionierendes Produkt wichtiger ist als eine perfekt ausgearbeitete Spezifikation.

Bestandteile des Scrum-Prozesses

Projektbeginn

Das Projekt beginnt mit einem Kick-off-Meeting. Dieses Meeting wird gemeinsam mit den Entwicklern und gegebenenfalls auch mit weiteren Teilnehmern wie etwa den Auftraggebern durchgeführt. Hier wird vor allem eine gemeinsame Vision entwickelt und die Anforderungen an das Produkt werden auf Story-Cards festgehalten.

Erstellung des Product Backlog

In Vorbereitung auf das nachfolgende „Sprint Planning“ legt der Product Owner ein sogenanntes Product Backlog an. Grundlage dafür bilden die zum Projektbeginn erstellten Story-Cards. Ein Product Backlog ist eine Liste mit den Anforderungen an das Projekt, welche während der Projektarbeit kontinuierlich vom Product Owner weiterentwickelt und angepasst wird. Dabei werden unter dem Aspekt der Kundenzufriedenheit die einzelnen Anforderungen mit Prioritäten versehen und gegebenenfalls unwichtige oder nicht zu realisierende Anforderungen aussortiert.

Planung der Sprints (Sprint Planning)

Ein Sprint ist ein Arbeitszyklus, in dem jedes Mitglied des Entwicklerteams eine von ihm ausgewählte Teilaufgabe bearbeitet. Die Planung der Sprints erfolgt in sogenannten Sprint-Planning- Meetings, bei denen der Product Owner und das Entwicklerteam zusammenkommen. Dieses Meeting besteht aus zwei Teilen, die jeweils ca. die Hälfte des Meetings beanspruchen. Im ersten Teil stellt der Product Owner dem Entwicklungsteam die Anforderungen aus dem Product Backlog vor, sodass sie dem Team verständlich sind. Daraufhin wird ein Sprintziel definiert, was einem Meilenstein entspricht und entschieden, welche Anforderungen (Backlog-Items) in einem Sprint umgesetzt werden. Im zweiten Teil des Meetings wird das Sprint Backlog angelegt.

Erstellung eines Sprint Backlog

Das Projektteam bespricht nun, wie die Backlog-Items aus dem Product Backlog in Einzel- oder Teilaufgaben (Task) zerlegt werden können. Die einzelnen Tasks werden dann im sogenannten Sprint Backlog festgehalten und an einem Board visualisiert. Für jeden Task muss festgelegt werden, wann dieser beendet ist (Definition of Done). Das Sprint Backlog stellt den Arbeitsplan für das Entwicklerteam dar. Hieraus wählt sich jedes Team- Mitglied gemäß seinen Kompetenzen ein Ticket und bearbeitet dieses anschließend.

Durchführung des Sprint und des täglichen Daily Scrum

Auf Basis des Sprint Backlogs wird vom Entwicklungsteam die tägliche Arbeit während des Sprints verrichtet, wobei jedes Teammitglied seine Aufgaben selbstständig bearbeitet. Ein Task ist fertig, wenn der Product Owner die „Fitness for Purpose“ erklärt hat, die Tests in Übereinstimmung mit den Abnahmekriterien erfolgreich waren und der Status einvernehmlich mit dem Entwicklerteam auf „Done“ gesetzt wurde. Einmal pro Tag führen die Entwickler außerdem ein fünfzehnminütiges Meeting (Daily Scrum) durch. Dabei werden die Veränderungen der letzten 24 Stunden besprochen. Zusätzlich geben alle Teilnehmer einen Ausblick auf die Tagesziele. Der Arbeitsfortschritt selbst wird in einem sogenannten Sprint-Burn-Down-Chart festgehalten. Der Scrum Master ist kein aktiver Teilnehmer des Meetings. Er stellt jedoch sicher, dass die Timebox eingehalten wird und das Meeting stattfindet.

Erstellung einer Impediment List

Es können immer wieder Schwierigkeiten bei der Durchführung des Projektes auftreten. Dazu zählen beispielsweise logistische und infrastrukturelle Hindernisse oder Probleme in der Kommunikation. Diese werden in einer sogenannten Impediment List erfasst. Die jeweils aktuellen Impediments werden vor allem im Rahmen der Daily-Scrum-Meetings angesprochen und vom Scrum Master dokumentiert. Ziel ist es, diese Hindernisse schnellstmöglich aus dem Weg zu räumen. Dazu initiiert der Scrum Master entsprechende Maßnahmen.

Durchführung einer Sprint Review

Kurz vor Ende des Sprints findet ein Sprint-Review-Meeting statt. Im Sprint Review präsentiert das Entwicklerteam dem Product Owner und allen interessierten Stakeholdern das Ergebnis der bisherigen Arbeit live am System und holt Feedback und Verbesserungsvorschläge ein. Auch das Product Backlog wird besprochen und notfalls angepasst.

Durchführung eines Retrospective-Meeting

Abschließend wird die Sprint Retrospektive abgehalten. Dabei handelt es sich um eine Art von „Lessons Learned“ (gewonnene Erkenntnisse), wobei sich das Scrum-Team selbst und die Arbeit im letzten Sprint reflektiert. Die Teilnehmer analysieren, was beibehalten werden sollte und was auf welche Weise verändert werden kann, um das Vorgehen noch effektiver zu gestalten. Dementsprechend wird die Definition of Done angepasst. Das Ziel besteht darin, die Produktqualität durch die Prozessqualität anzuheben. Darauf folgen weitere Zyklen, in denen das Produkt vervollständigt wird. Das Treffen wird vom Scrum Master moderiert.

Durchführung von weiteren Sprints

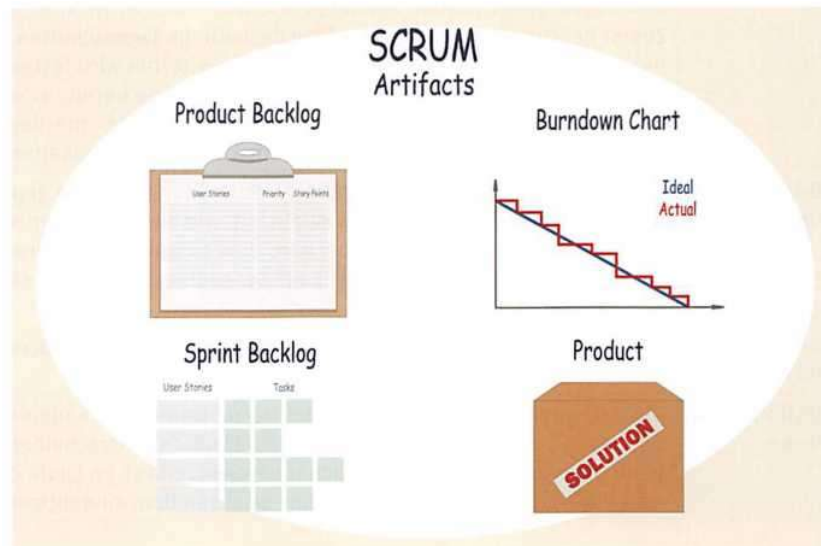
Sind die Ziele für den entsprechenden Entwicklungsabschnitt des Produktes erfüllt, dann beginnt wieder ein neuer Sprintzyklus und der Product Owner und das Entwicklungsteam treffen sich zur Durchführung des nächsten Sprint Plannings. Dieser Ablauf wiederholt sich so lange, bis das Projekt beendet werden kann.

Projektende

Ist das Produkt fertiggestellt und vom Product Owner abgenommen worden, kann das Projekt beendet werden. Mit dem finalen Projektabschluss sollte noch ein Project Review erfolgen. Hier werden alle Aspekte angesprochen, welche die Zusammenarbeit erschwert oder den Prozess behindert haben. Daraus ergeben sich meistens viele Möglichkeiten, das nächste Scrum-Projekt weiter zu verbessern.

Artefakt

In Scrum stellen sogenannte Artefakte (vom Menschen erzeugte Produkte) die Transparenz und das gemeinsame Verständnis von dem Projekt und der zu erstellenden Software sicher. Dabei sind folgende „Dokumente“ vorgeschrieben: Product Backlog, Sprint Backlog, Product Increment, Definition of Done. Insbesondere beim Product Increment handelt es sich dabei nicht um ein klassisches Dokument, sondern um das Softwareprodukt. Deshalb wird auch der Begriff „Artefakt“ gewählt.



Artefakte in Scrum			
Product Backlog	Das Product Backlog ist eine geordnete Liste aller Anforderungen an das Produkt. Im Gegensatz zu einem Lasten- oder Pflichtenheft erlangt es bis zum Projektende keine Vollständigkeit. Es ist ein dynamisches Dokument und die einzige Quelle, worin alle Anforderungen, Funktionalitäten, Eigenschaften und Änderungen festgehalten und priorisiert werden. Die jeweiligen Einträge nennen sich Items und werden typischerweise mit einer Beschreibung, Priorisierung und Aufwandsschätzung versehen. Die Ordnung der Items erfolgt nach Ermessen des Product Owners, was beispielsweise anhand der Priorität, des Risikos oder der Notwendigkeit einzelner Items erfolgen kann. Die Priorität von Einträgen im Product Backlog nimmt nach unten hin ab. Ein einfaches Product Backlog könnte wie folgt aussehen:		
	Priorität	Beschreibung	Aufwand
	1	Aufgabenbeschreibung 1	5
	2	Aufgabenbeschreibung 2	14
	3	Aufgabenbeschreibung 3	1
	4	Aufgabenbeschreibung 4	7
	5	Aufgabenbeschreibung 5	2
	Wert		
			20
			80
			30
			25
			25
	Ganz oben stehen die „User-Stories“, die konkret und klein genug für eine Umsetzung im Sprint sind. Weiter unten stehen die „Epics“ mit geringerer Priorität oder höherer Komplexität, die erst für nächste Sprints infrage kommen oder zu groß sind, um in einem Sprint umgesetzt zu werden.		
Sprint Backlog	Für jeden Iterationszyklus existiert ein sogenanntes Sprint Backlog. Dieses Backlog beinhaltet eine ausgewählte Menge an Stories, die in dem aktuellen Zyklus abgearbeitet werden sollen. Die Aufgaben werden nach ihrer Priorität in das Sprint Backlog aufgenommen und abgearbeitet. Dazu wählt das Entwicklungsteam nur hochpriorisierte Items aus dem Product Backlog aus. Über die Menge der Aufgaben, die in einer Iteration abgearbeitet werden können, entscheidet das Entwicklungsteam. Zuerst beschreibt das Sprint Backlog deshalb die Eigenschaften (Story), die das Produkt am Ende einer Iteration besitzen soll. Weiterhin wird festgehalten, welche Aufgaben noch zu erledigen, welche zu testen und welche bereits erledigt sind. Daher ist auch das Sprint Backlog kein statisches Dokument und unterliegt ständiger Veränderung. So wird der Status des Sprints immer transparent gehalten.		
Burn-down-Chart	Ein Burn-Down-Chart ist eine grafische Darstellung über die geplante und die geleistete Arbeit in einem Projekt. Es visualisiert, wie viel Arbeit zum aktuellen Zeitpunkt noch erledigt werden muss. Dabei wird die Zeit auf der waagerechten und die zu erledigende Arbeit („Sprint Backlog“) auf der senkrechten Achse dargestellt (siehe auch S. 159).		
Product Increment	Ein Increment ist in Scrum das Ergebnis einer Iteration. Ein Increment muss am Ende einer Iteration potenziell einsatz- bzw. releasefähig sein.		
Definition of Done	Die Definition of Done beschreibt das gemeinsame Verständnis des Teams darüber, was „done“, also fertig, bedeutet. Diese Definition unterscheidet sich je nach Scrum-Team und verändert sich zusätzlich im Projektverlauf. Im Laufe des Scrum-Prozesses wird diese Definition in ihrem Umfang wachsen bzw. eindeutiger und genauer werden.		

In Scrum und im agilen Projektmanagement im Allgemeinen ist es üblich, dass Plan und Umsetzung mit der Zeit voneinander abweichen. Deshalb müssen agile Teams Kontrolle über ihren

geplanten und ihren tatsächlichen Projektfortschritt behalten. Dazu hat sich ein Burn-down-Chart als visuelles Planungs-, Kontroll- und Steuerungswerkzeug bewährt. Allerdings gibt es keine genauen Vorschriften, ob und wie ein Burn-down-Chart zur Fortschrittsüberwachung zu benutzen ist. Der Umfang der dargestellten Aufgaben sollte allerdings nicht zu umfangreich sein, damit der Restaufwand realistisch eingeschätzt werden kann.

Burn-down-Chart bezogen auf Scrum-Projekte

Aufbau

X-Achse: Zeitverlauf, Y-Achse: verbleibender Arbeitsaufwand in Stunden oder in Aufgaben

Mögliche Kurvenverläufe

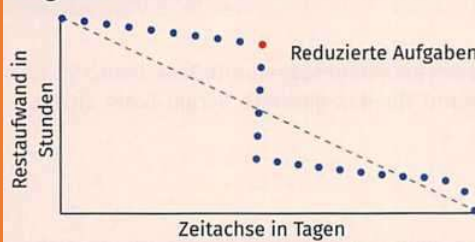
Idealer Verlauf



Tatsächlicher Verlauf ohne Veränderungen der Aufgaben



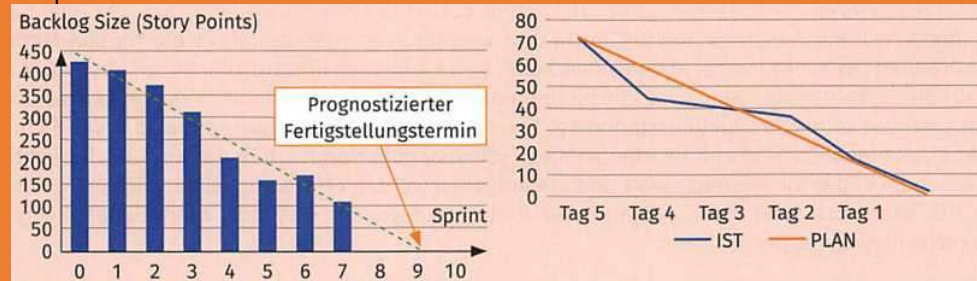
Tatsächlicher Verlauf mit Reduzierung von Aufgaben



Tatsächlicher Verlauf mit zusätzlichen Aufgaben



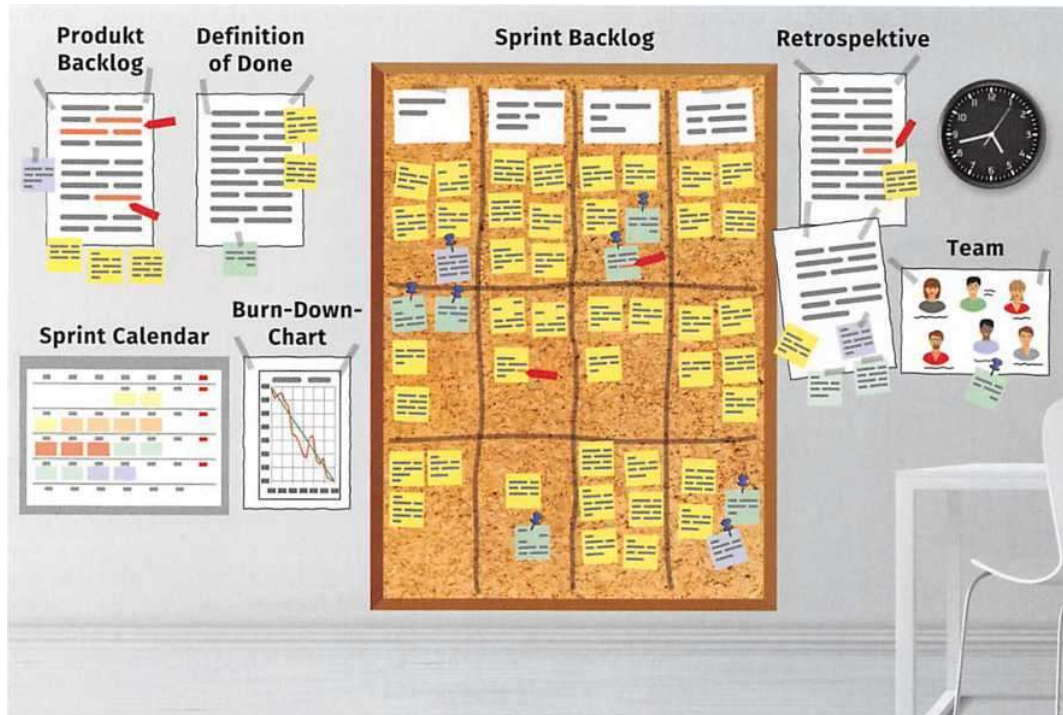
Beispiele



Vorteile

- Sehr einfach und verständlich
- Ermöglicht eine effektive Kontrolle über den Fortschritt im Sprint
- Gibt eine schnelle Übersicht über die noch zu leistende Arbeit
- Sehr gutes Mittel zur Kommunikation mit Kunden und Stakeholdern
- Guter Indikator für die Effizienz der Teamarbeit
- Wichtiges Mittel zum Abschätzen von Risiken

Task-Board



Alle Bereiche und Aufgaben eines Scrum-Projektes werden an einem sogenannten Task-Board visualisiert. Dadurch werden die Tasks und deren Bearbeitungsstand für das gesamte Scrum-Team sichtbar und transparent.

LE 4 - Kanban

Neben Scrum ist Kanban eine weitere weit verbreitete agile Projektmanagementmethode. Dabei wird eine Gesamtaufgabe in einzelne Schritte zerlegt und diese werden einzeln abgearbeitet. Damit wird das Risiko für jede einzelne Maßnahme reduziert, kürzere Bearbeitungszeiten und eine bessere Qualität erreicht. Dieses war auch die Intention für Taiichi Ōno, welcher in den 1950er-Jahren die Kanban-Methode für die Produktionssteuerung bei Toyota entwickelte. David Anderson übertrug die Prinzipien von Kanban 2007 auf die Softwareentwicklung. Im Prinzip funktioniert die Kanban-Methode in der Softwareentwicklung ebenso wie in der Produktionssteuerung: Teammitglieder eines Softwareprojekts holen sich ihre Aufgaben aus einem Backlog und bearbeiten diese selbstständig. Somit erhält das einzelne Projektteam mehr Autonomie, die Transparenz steigt und die Aufgaben werden effizienter gelöst. Mit der Zeit entwickelte sich Kanban zu einer vollständigen agilen Projektmanagement-Methode, welche unabhängig von der Branche verwendet werden kann.

Die grundlegende Idee bei der Anwendung der Kanban-Methode besteht darin, die vorhandene Arbeit in einzelne Aufgaben zu zerlegen und diese und deren Bearbeitungsstand zu visualisieren. Dieses wird mithilfe eines Kanban-Boards realisiert. Klassisch wird ein Kanban-Board in drei Spalten eingeteilt und wie folgt benutzt:

1. To Do: Hier werden die einzelnen Aufgaben, welche z.B. auf kleinen Karten notiert und deren Bearbeitung noch nicht begonnen wurde, hinzugefügt.
2. In Progress/Doing: Sobald die Bearbeitung einer Aufgabe beginnt, wird die entsprechende Karte in diese Spalte verschoben.
3. Done: Ist die Aufgabe erledigt, wandert die Karte in diese Spalte.



Da Kanban ein sehr flexibler Ansatz ist, können sowohl die Anzahl der Spalten als auch deren Überschriften an die Anforderungen des Projekts angepasst werden. In manchen Fällen eignen sich beispielsweise vier- oder fünfspaltige Varianten besser, um alle Stufen des Entwicklungsprozesses zu erfassen. Eine einfache Umsetzung besteht aus dem Einsatz eines Whiteboards und Haftnotizen. Kanban-Boards können aber auch digital geführt werden. Aber auch eine kombinierte Nutzung von digitalen Tools und physischem Board ist durchaus üblich. Als nächstes beginnt der Workflow. Dabei wählen die einzelnen Teammitglieder sich selbstständig Aufgaben aus der ersten Spalte aus, fügen sie der mittleren Spalte hinzu und beginnen mit der Arbeit. Ist die Aufgabe erledigt, dann fügen sie diese der rechten Spalte hinzu. Dabei wird angestrebt, dass möglichst alle Aufgaben gleichmäßig, ohne Verzögerungen und Blockaden abgearbeitet werden. Allein das Kanban-Board führt zu viel Transparenz über die Verteilung der Arbeit sowie bestehende Engpässe. Stockt der Entwicklungsprozess an einer Stelle, ist dieses sehr schnell zu erkennen und es können entsprechende Maßnahmen getroffen werden. Wie jedes Vorgehensmodell im Projektmanagement besitzt auch Kanban seine Vor- und Nachteile.

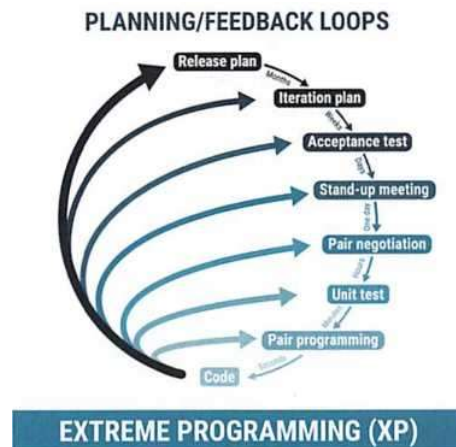
Vor- und Nachteile von Kanban	
Vorteile	Nachteile
- Gibt schnell und zuverlässig eine Übersicht über den Projektfortschritt und akute Probleme - Einzelne Arbeitsschritte werden schneller bearbeitet - Reduziert den Managementaufwand - Lässt sich auch sehr gut außerhalb der Softwareentwicklung für andere Bereiche, wie Wartung oder Systemadministration einsetzen - Kann sehr gut mit anderen Managementmethoden kombiniert werden - Lässt sich sehr schnell und problemlos einführen - Erhöht die Motivation der Mitarbeiter	- Softwareentwickler müssen über breite fachliche Fähigkeiten verfügen, für den Fall, dass ein Teammitglied ausfällt oder überlastet ist - Zu lösendes Problem muss sich in einzelne voneinander abgrenzbare Einzelaufgaben einteilen lassen - Gutes Zeitmanagement lässt sich schlechter realisieren als in anderen Vorgehensmodellen - Für größere Projekt eher ungeeignet

Im Vergleich mit Scrum weisen beide Vorgehensmodelle einige Gemeinsamkeiten auf, z. B. die agile Vorgehensweise, transparente Prozesse oder dass die Teammitglieder ihre Aufgaben aus einem Backlog auswählen. Es gibt aber auch einige Unterschiede. Allgemein kann festgestellt werden, dass Scrum das umfassendere Vorgehensmodell ist. Kanban besitzt auch einen theoretischen Hintergrund, lässt sich aber sehr leicht nur auf die visuelle Darstellung des Aufgabenmanagements mittels eines Kanban-Boards reduzieren. Dadurch kann Kanban sehr flexibel mit anderen Methoden kombiniert werden.

Vergleich	
Kanban	Scrum
- wenige feste Regeln - sowohl für Projekte als auch Routineaufgaben geeignet - ab einer gewissen Projektgröße besteht die Gefahr, dass Kanban unübersichtlich wird - kennt keine festen Rollen - Priorisierung im Backlog optional - Iterationen sind optional	- komplexes Regelwerk - nur für Teams ab drei Personen geeignet - nur für Projekte, nicht für Routineaufgaben geeignet - Scrum-Board wird nach jedem Sprint neu mit Aufgaben versehen - feste Rollen (Product Owner, Scrum Master, Team) - Priorisierung im Backlog vorgeschrieben - Iterationen sind vorgeschrieben

LE 5 - Extreme Programming (XP)

Extreme Programming ist ein weiteres Vorgehensmodell der agilen Softwareentwicklung. Dabei stehen kurze Entwicklungszyklen und schnelle Reaktionszeiten auf neue oder sich ändernde Anforderungen im Vordergrund. Anders als bei anderen Vorgehensmodellen gibt es am Anfang keinen strikt definierten Anforderungskatalog und das zu entwickelnde System wird nicht im Detail entworfen, sondern es wechseln sich ständig Entwurfs- und Entwicklungsphasen ab. Je mehr die Entwicklung der Software voranschreitet, desto mehr Kundenwünsche werden in diese integriert. Die einzelnen Entwicklungszeiten reichen dabei von einem Tag bis zu mehreren Wochen. Regelmäßige Test bei der Entwicklung sollen dabei ein frühes Entdecken von Fehlern ermöglichen. Das Extreme Programming wurde von dem Softwareentwickler Kent Beck entwickelt und erstmals in der Mitte der 1990er-Jahre im Chrysler-Comprehensive-Compensation-Programm (CS-Programm) erfolgreich eingesetzt. Allerdings kann XP nicht für alle Arten von Projekten eingesetzt werden. Es eignet sich vor allem für Projekte, bei denen die Anforderungen an die Software schnell wachsen und ständigen Veränderungen unterworfen sind, und für kleine Entwicklungsteams, welche eng zusammenarbeiten können.



Rollen im Extreme Programming

Kunde

Der Kunde ist Auftraggeber und er entscheidet über die Funktionalität der Software. Dazu wählt er aus, was in welcher Reihenfolge umzusetzen ist. Darüber hinaus definiert er die Tests, welche die Funktionalität des Systems überprüfen und nimmt diese ab.

Entwickler

Der Entwickler setzt die Anforderungen des Kunden um. Dazu schätzt er den Aufwand und die Komplexität der User Stories ab und steuert die Geschwindigkeit, mit der die Funktionalität entwickelt und ausgeliefert wird.

Manager

Der Manager bringt die Kunden und Entwickler zusammen. Im Laufe des Entwicklungsprozesses unterstützt er beide Seiten und hilft ihnen, zu einem schlagkräftigen Team zusammenzuwachsen.

Wie jedes Vorgehensmodell im Projektmanagement besitzt auch das Extrem-Programming Vor- und Nachteile.

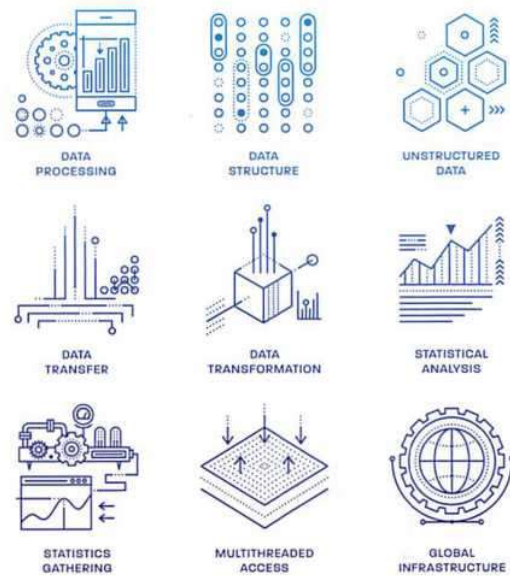
Vor- und Nachteile von Extreme Programming	
Vorteile	Nachteile
• Problemorientierte Entwicklung • Komplexe Softwareprojekte werden beherrschbar • Kürzere Entwicklungszeiten • Einsparung von Zeit und Kosten • Aktive Einbindung des Kunden • Relativ einfache Einzelschritte • Zeitnahe Integration von Kundenwünschen • Ermöglicht eine hohe Qualität der Software • Gewährleistet eine gute Wart- und Erweiterbarkeit der Software	• Für sehr kleine Softwareprojekte eher ungeeignet • Sehr hohe Anforderungen an die Fachlichkeit und die Qualität der Entwickler • Kunden haben meist nicht ausreichend Zeit, um sich in der geforderten Form einzubringen • Programmierer werden dazu verleitet, Aufgaben sehr schnell fertigzustellen, was teilweise in dem Auslassen des Testens endet • Kunden müssen Vertrauen in die Kompetenz und Kontinuität der Entwickler haben

Manchmal wird das Extreme Programming mit Scrum gleichgesetzt. Dies ist nicht korrekt, da beide Methoden unabhängig voneinander bestehen. Allerdings passen beide Methoden sehr gut zusammen. So kann XP gut in Scrum integriert werden.

Vergleich der Methoden	
Extreme Programming	Scrum
• Iterationen dauern i. d. R. eine oder zwei Wochen. • Solange noch nicht mit der Abarbeitung einer Aufgabe begonnen wurde, kann diese noch durch eine andere (wichtigere) ersetzt werden. • Die Zusammensetzung des Entwicklerteams kann sich während einer Iteration auch ändern und flexibel angepasst werden. • Aufgaben werden in einer strengen Prioritätsreihenfolge abgearbeitet, welche vom Auftraggeber (Kunden) festgelegt wird. • Die Methode schreibt technische Praktiken wie testgetriebene Entwicklung, einfaches Design oder Pair Programming vor.	• Iterationen (Sprints) dauern normalerweise zwei Wochen bis zu einem Monat. • Es gibt i.d.R. keine Änderungen bei den Sprints mehr, wenn das Planungsmeeting abgeschlossen ist. • Sprint-Teams bleiben während eines Sprints unverändert. • Der Product Owner priorisiert die Aufgaben. Das Entwicklerteam bestimmt, wann abgearbeitet wird. Aber auch hier wird i. d. R. an den Aufgaben mit der höchsten Priorität gearbeitet (muss aber nicht). • Die Methode schreibt keine technischen Praktiken vor.

TAG 2 - Daten bewerten und aus verschiedenen Datenquellen zusammenführen

In vielen Unternehmen werden große Mengen an Daten erzeugt. Die Informationen, welche in diesen Daten stecken, sind ein entscheidender Faktor im Unternehmenswettbewerb. Die Daten selbst stammen i.d.R. aus vielen unterschiedlichen (heterogenen) Datenquellen, z. B. Daten aus der Kommunikation mit Kunden, Sensordaten von Maschinen oder Daten aus finanziellen Transaktionen. Diese Vielfalt an Datenquellen stellt eine nicht zu unterschätzende Herausforderung für die Unternehmen dar, da sie i.d.R. nicht über einen zentralen Zugriffspunkt auf diese Daten verfügen. Die Integration heterogener Datenquellen ist deshalb eine wichtige Anforderung an neue Softwareprodukte. Dabei spielt die Bewertung der Daten nach bestimmten Qualitätskriterien eine immer größere Rolle.



LE 1 - Die Qualität von Daten beschreiben

Die Qualität von Daten wird i.d.R. als die Eignung der Daten für die jeweilige datenverarbeitende Anwendung angesehen. Daten von schlechter Qualität enthalten beispielsweise Datenfehler, Dubletten, fehlende Werte, falsche Formatierungen oder Widersprüche. Da Informationen mithilfe von Daten in IT-Systemen abgebildet werden, wirkt sich die Datenqualität auch auf die Qualität der Informationen aus, welche aus diesen Daten gewonnen werden.



Es gilt die Regel: Es sind keine guten Informationen aus schlechten Daten zu gewinnen.

Datenqualität

Datenqualität ist ein Maß für die Eignung von Daten für Befriedigung von Bedürfnissen nach konkreten Informationen. Anhand von verschiedenen Kriterien lässt sich die Qualität eines Datenbestands beurteilen.

Für Unternehmen kann die Datenqualität entscheidend für den geschäftlichen Erfolg sein. Durch Datenbestände mit geringer Qualität und fehlerhaften Daten können den Unternehmen erhebliche Kosten entstehen. So verursachen etwa Doppelungen bei Kundenadressen einen erhöhten finanziellen und zeitlichen Aufwand beim Versand von Produkten. In der Industrie und den Naturwissenschaften spielt z. B. die Qualität von Daten, welche aus Messungen gewonnen werden, eine große Rolle. Je weniger Störeinflüsse die Messung beeinflussen, je präziser die

Messung ist und umso mehr Messungen vorgenommen werden, desto besser ist die Datenqualität. Auch Nachrichtendienste brauchen Daten von möglichst guter und sehr guter Qualität. Hier müssen z. B., um Irrtümer und Fehlinformationen auszuschließen, die Daten immer wieder anhand anderer Quellen überprüft werden. In der Wirtschaft wiederum werden auf der Basis von Daten und den daraus gewonnenen Informationen zum Teil weitreichende Entscheidungen gefällt, Marktchancen bewertet oder Verhandlungen geführt. Deswegen ist auch in diesem Bereich die Qualität der Daten von entscheidender Bedeutung.

Beispiele für mangelnde Datenqualität



Schon bei der Erfassung von Daten kann die Datenqualität aufgrund unterschiedlicher (heterogener) Datenquellen stark beeinflusst werden. Aber auch die Speicherung und Verwaltung der Daten hat großen Einfluss auf die Qualität der Daten.

Es gibt eine große Zahl von Qualitätskriterien, nach denen Daten beurteilt werden können. Wie wichtig die einzelnen Qualitätskriterien sind, hängt i.d.R. vom Nutzer und dessen Bedürfnissen ab. Insofern ist es sehr wichtig für den Softwareentwickler zu wissen, wie der Nutzer (Auftraggeber) der zu entwickelnden Software die anzusetzenden Qualitätskriterien für die Daten beurteilt.

Von den aufgeführten Qualitätskriterien werden beinahe nie alle gleichzeitig zur Beurteilung der Daten herangezogen. Vielmehr wird eine Auswahl getroffen, welche den Kundenanforderungen am besten gerecht wird. Diese sollte am Anfang auch nicht zu viele Qualitätskriterien umfassen. Im Laufe der Entwicklung der Software können später immer noch weitere Kriterien hinzukommen.

Qualitätskriterien für Daten		
Qualitätskriterium	Beschreibung	Beispiel
Vollständigkeit (Completeness)	Ein Datensatz ist vollständig, wenn alle Attribute gültige Werte enthalten. Die Vollständigkeit wird in Prozent angegeben.	Ein vollständiger Datensatz für eine Person muss folgende Daten enthalten: ID, Vorname, Nachname, Geburtsdatum. Fehlt einer oder fehlen mehrere Werte, dann ist der Datensatz unvollständig. Fehlt etwa bei 15 von 100 Datensätzen das Geburtsdatum, dann beträgt die Vollständigkeit der Datensätze 85 %.

Eindeutigkeit (Uniqueness)	Jeder Datensatz muss eindeutig interpretiert werden können. Die Eindeutigkeit wird in Prozent angegeben.	Zwei Datensätze (mehrere Attribute) von Personen weisen bis auf ein Attribut (z. B. Geburtsdatum) die gleichen Werte auf. Es ist davon auszugehen, dass es sich dabei um eine „Dublette“ handelt. Allerdings ist das Geburtsdatum in diesem Fall nicht eindeutig zu interpretieren.
Korrektheit (Correctness)	Die Daten müssen mit der Realität übereinstimmen. Die Korrektheit wird in Prozent angegeben.	Das Geburtsdatum einiger Personen (nur lebende Personen werden verwaltet) enthält den Wert 01.01.1900. Im Jahr 2022 kann man davon ausgehen, dass der Wert falsch ist. Wenn bei 3 von 100 Datensätzen der 01.01.1900 als Wert eingetragen ist, dann beträgt die Korrektheit der Datensätze 97%.
Aktualität (Timeliness)	Alle Werte der Datensätze müssen jeweils den aktuellen Zustand der Realität abbilden. Die Aktualität wird in Prozent angegeben.	Es werden die Daten von 100 Personen (Name, Adresse etc.) in der Datenbank gespeichert. Da Personen im Laufe der Zeit umziehen oder heiraten, verlieren auch die gespeicherten Daten an Aktualität. Durch entsprechende Maßnahmen muss von Zeit zu Zeit die Aktualität überprüft werden. Wenn dabei festgestellt wird, dass von 100 Personen bei 23 die Daten nicht mehr aktuell sind, dann beträgt die Aktualität der Daten 77%.
Genauigkeit (Accuracy)	Die Daten müssen in der jeweils geforderten Exaktheit vorliegen. Die Genauigkeit wird in Prozent angegeben.	Die Messdaten einer Maschine sollen immer mit vier Nachkommastellen abgespeichert werden. Zwei von 100 Messdaten weichen von dieser Regel ab. Die Genauigkeit beträgt in diesem Fall 98%.
Konsistenz (Consistency)	Ein Datensatz darf in sich und zu anderen Datensätzen keine Widersprüche aufweisen. Die Konsistenz wird in Prozent angegeben.	In einem Datensatz über verkaufte Artikel wird neben der Anzahl der Artikel und dem Preis pro Artikel auch noch der Gesamtbetrag gespeichert. Wenn in einem Datensatz dann folgende Werte stehen: Anzahl: 20, Preis: 20.00, Gesamt: 500.00 dann ist der Datensatz in sich widersprüchlich.
Redundanzfreiheit (Nonredundant)	Es dürfen keine Informationen mehrfach vorhanden sein. Die Redundanzfreiheit wird in Prozent angegeben.	Aus verschiedenen Datenquellen wurden redundant vorgehaltene Kundendaten zusammengeführt. Bei Auswertungen wurde festgestellt, dass z. B. einige Kundennummer doppelt vorkommen. Damit sind diese Datensätze nicht redundanzfrei.

Qualitätskriterien für Daten		
Qualitätskriterium	Beschreibung	Beispiel
Relevanz (Relevancy)	Der Informationsgehalt von Datensätzen muss den jeweiligen Informationsbedarf erfüllen. Die Relevanz wird in Prozent angegeben.	Es sollen die Umsätze eines bestimmten Quartals analysiert werden. Alle Datensätze, welche nichts mit den Umsätzen des entsprechenden Quartals zu tun haben, sind für die Auswertung nicht relevant. Wenn von 100 übermittelten Umsätzen 11 nichts mit dem Quartal zu tun haben, dann besitzen die entsprechenden Datensätze eine Relevanz von 89%.
Einheitlichkeit (Uniformity)	Gleiche Werte eines Datenbestandes müssen einheitlich strukturiert sein. Die Einheitlichkeit wird in Prozent angegeben.	Das Datum ist immer im Format der ISO 8601 abzulegen (YYYY-MM-DD). Wenn von 100 000 Datumsangaben 2 000 in einem anderen Formatvorliegen (z. B. DD.MM.YYYY), dann liegt bei den Datumsangaben eine Einheitlichkeit von 98% vor.
Zuverlässigkeit (Reliability)	Die Entstehung der Daten muss nachvollziehbar sein. Die Zuverlässigkeit wird in Prozent angegeben.	Es werden Daten aus mehreren heterogenen Datenquellen zusammengeführt. Dabei kommen 94% der Daten aus verlässlichen Quellen, d. h., die Entstehung der Daten und deren Qualität ist seit Jahren bekannt. Der Rest der Daten kommt aus Quellen, deren Datenqualität noch nicht genau bekannt ist und auch größeren Schwankungen unterliegen kann. Die Zuverlässigkeit der zusammengeführten Daten beträgt in diesem Fall 94%.
Verständlichkeit (Understandability)	Die Datensätze müssen in ihrer Begrifflichkeit und Struktur mit den Vorstellungen der Informationsempfänger (z. B. Fachbereiche) übereinstimmen. Die Verständlichkeit wird in Prozent angegeben.	Daten sind dann verständlich, wenn sie unmittelbar vom Benutzer verstanden und für seine Zwecke eingesetzt werden können. Wenn beispielsweise ein Kunde Daten für ein Mobiltelefon abrufen, dann sind die Angaben zum Hersteller, zur Farbe und zur Speichergröße deutlich verständlicher als eine entsprechende Referenz- bzw. Modellnummer (z. B. AI305).
Zugänglichkeit (Accessibility)	Die Daten müssen bei Informationsbedarf auf einfachem Wege direkt zugänglich und verfügbar sein. Die Zugänglichkeit wird in Prozent angegeben.	Die Bestelldaten und die Sendungsverfolgung müssen beispielsweise für einen Kunden 24 Stunden am Tag über verschiedene Medien abrufbar sein. Der Abruf der Daten sollte dabei nicht viel Zeit in Anspruch nehmen.

Für eine Anwendung sind wie erwähnt nicht alle Kriterien von gleicher Bedeutung und es gilt, erst einmal die wichtigsten Kriterien festzulegen und anhand dieser die Qualität der Daten zu bewerten. Dazu sollten die Anforderungen an die Datenqualität nach Bedeutung gewichtet betrachtet werden. Sind die Kriterien erst einmal festgelegt, dann sollte eine genaue Analyse der Daten unter diesen Gesichtspunkten erfolgen. Die Qualität der Daten, selbst bei einfachen Anwendungen, nur aus dem Gefühl heraus zu beurteilen, ist in keinem Fall zielführend und führt in vielen Fällen zu einer falschen Beurteilung der Qualität. Ist erst einmal ein hohes Level bezüglich der Datenqualität erreicht, gilt es, dieses zu bewahren. Dazu sollten die Datenbestände regelmäßig analysiert und bereinigt werden.

Prozess zur Verbesserung der Qualität von Daten

Analyse der Daten (Data Profiling)

Hier wird versucht, Fehler und Widersprüche in den Datenbeständen zu erkennen. Dies können z. B. inkonsistente oder fehlerhafte Daten sein. Aus den Ergebnissen dieser Analyse werden die weiteren Maßnahmen zur anschließenden Bereinigung der Daten abgeleitet.

Bereinigung der Daten (Data Cleaning)

In dieser Phase werden die bei der Analyse erkannten Fehler und Widersprüche beseitigt. Dabei kommen unterschiedliche Verfahren und Algorithmen zur Anwendung. Beispiele für die Datenbereinigung sind beispielsweise die Beseitigung von Dubletten oder die Vervollständigung von Datensätzen.

Monitoring der Daten (Data Monitoring)

Nachdem durch die Bereinigung der Daten eine gewisse Datenqualität erreicht wurde, wird nach einer gewissen Zeit durch ein Monitoring diese erneut überprüft. Dadurch lässt sich die Qualität der Datenbestände über größere Zeiträume bewahren. Fehlt dagegen ein Monitoring der Daten, sinkt die erreichte Qualität mit der Zeit kontinuierlich ab (siehe auch Datenalterung). Deswegen muss das Monitoring in regelmäßigen Abständen wiederholt werden.

Für kleinere Datenbestände kann dieser Prozess noch manuell durchgeführt werden. Der benötigte Personal- und Zeitaufwand steigt aber mit zunehmender Datenmenge schnell an und ist im Big-Data-Bereich mit seinen riesigen Datenbeständen manuell nicht mehr zu bewerkstelligen. Deswegen gibt es auf dem Markt zahlreiche Anwendungen, mit deren Hilfe der Aufwand zur Sicherstellung einer hohen Datenqualität, durch Automatisierung der oben genannten Prozesse, minimiert werden kann.

Datenalterung

Datenbestände unterliegen Veränderungen. So ändern sich mit der Zeit beispielsweise Adressen von Personen oder die Preise von Artikeln. Über einen längeren Zeitraum betrachtet verschlechtert sich dadurch nach und nach die Qualität des Datenbestandes. Diesen Vorgang nennt man auch Datenalterung.

Ein systematisches Datenqualitätsmanagement trägt dauerhaft dazu bei, eine hohe Datenqualität zu erhalten. Dieses muss genauso ins Gesamtqualitätsmanagement eines Projektes integriert werden wie ein Informationssicherheitsmanagement.

TAG 3 - Daten aus Datenquellen abrufen

Wenn eine Software für einen Kunden entwickelt werden soll, welche Daten verarbeitet und ausgewertet, dann können diese Daten aus externen Datenquellen stammen oder vom Programm selbst generiert werden. Die Gewinnung der Daten aus externen Quellen kann beispielsweise durch das Abfragen von Daten aus Datenbanken, durch die Eingabe der Daten durch die Benutzer oder auch durch Echtzeitübermittlung von Maschinen geschehen. Allgemein kann festgestellt werden, dass die Daten für Programme i.d.R. aus einer oder mehreren Datenquellen stammen. Diese werden dann vom Programm weiterverarbeitet. Zunächst wird der Begriff Datenquelle kurz näher betrachtet.



Datenquelle

Eine Datenquelle (Data Source) ist der Ort, von dem die Daten stammen, welche verwendet werden. Das kann der Ort sein, an dem die Daten entstanden sind (primäre Datenquelle) oder der Ort, wo sie für eine weitere Verwendung gespeichert wurden (sekundäre Datenquelle).

Beispiele für Datenquellen im IT-Bereich sind Datenbanken, einfache Dateien, Maschinen, welche digitale Echtzeitdaten generieren oder Videokonferenzen. Heutzutage erhöht sich, bedingt durch das Internet of Things (IoT) oder Big Data, rasant die Menge und die Vielfalt der erfassten Daten. Damit verbunden steigen gleichzeitig die Anzahl und die Artenvielfalt von zur Verfügung stehenden Datenquellen.

Die Daten aus den verschiedenen Datenquellen lassen sich in offene (Open Data) und geschlossene Daten (Closed Data) einteilen. Open Data sind Daten, welche uneingeschränkt genutzt werden können. Wenn es Einschränkungen geben sollte, dann betreffen diese beispielsweise die Nennung des Urhebers oder die Verwendung einer Share-alike-Klausel. Das Konzept von offenen Daten ist an Konzepte wie Open Access oder Open Source angelehnt. Dadurch, dass diese Daten z. B. von Bürgern, Nichtregierungsorganisationen, Hochschulen, Firmen usw. uneingeschränkt genutzt und verarbeitet werden können, können wiederum neue Erkenntnisse, Anwendungen, Produkte und Dienstleistungen usw. entstehen.

Beispiele für Open-Data-Quellen

Berlin Open Data, Land Schleswig-Holstein, Open-Government-Data-Portal, Rheinland-Pfalz, GDI-DE, Freistaat Sachsen, Open.NRW, Statistisches Bundesamt, Transparenzportal Hamburg, Regionalstatistik, Bundesministerium für Arbeit und Soziales, OpenData.HRO, mCLOUD, Wikipedia, Wikidata, NASA, CERN, DBpedia, European Union Open Data Portal, Google Public Data Explorer, Open Science Data Cloud, Climate Data Online

Das Gegenteil von Open Data sind Closed Data. Hier unterliegen die Daten verschiedenen Einschränkungen bezüglich der Zugänglichkeit, der Möglichkeit diese maschinell zu verarbeiten, den Kosten oder den zugrundeliegenden Rechten. Dabei ist der Übergang von Open Data zu Closed Data fließend und es gibt viele Zwischenformen, welche u.a. durch die oben genannten Eigenschaften bestimmt werden. So kann es z. B. Daten geben, welche öffentlich zugänglich sind, aber nur in analoger Form vorliegen, was eine maschinelle Verarbeitung erschwert.

Vergleich von Open Data mit Closed Data unter verschiedenen Gesichtspunkten		
Open Data (vollständig offen) → Closed Data (vollständig geschlossen)		
Zugänglichkeit	Jeder Mensch hat Zugang.	Nur bestimmte Menschen bzw. Organisationen haben Zugang.
Möglichkeit, die Daten maschinell zu verarbeiten	Die Daten liegen in Formaten vor, welche sehr leicht maschinell verarbeitet werden können.	Die Daten sind nur sehr schwer maschinell zu verarbeiten (z. B. analoge Daten).
Kosten	Die Nutzung der Daten verursacht keine Kosten.	Die Nutzung der Daten verursacht erhebliche Kosten.
Rechte	Daten können unlimited verarbeitet und weitergegeben werden.	Die Nutzung und Verbreitung der Daten sind verboten.

Die offene Nutzung von Daten wird nach wie vor kontrovers diskutiert. Dabei werden folgende Argumente ins Feld geführt.

Argumente für und gegen Open Data	
Argumente dafür (Pro) - Die Daten gehören den Menschen und nicht irgendwelchen Konzernen. - Daten, welche mithilfe von öffentlichen Geldern gewonnen wurden, müssen auch öffentlich zugänglich sein. - Fakten unterliegen nicht dem Urheberrecht. - Es können mehr Menschen die Daten nutzen und somit entsteht theoretisch auch mehr Innovation.	Argumente dagegen (Kontra) - Es besteht die Möglichkeit des finanziellen Verlusts für Daten, von denen man der Urheber ist. - Infragestellung der Qualität der Daten: „Was nichts kostet, ist nichts wert“. - Offene Daten verzerren den Wettbewerb (Wettbewerbsrecht). - Missbrauch von Daten wird erleichtert, z. B. für unmoralische Forschungsprojekte. - Es bestehen datenschutzrechtliche Bedenken. - Haftungsfragen bei öffentlichen Daten im Schadensfall sind unklar.

LE 1 - Datenformate zur Datenübertragung

Zum Austausch und zur Speicherung von Daten werden Dateien verwendet. Dabei werden die Daten in einem bestimmten Format in der Datei abgelegt, welches die innere Struktur und den Inhalt der Daten einer Datei beschreibt.

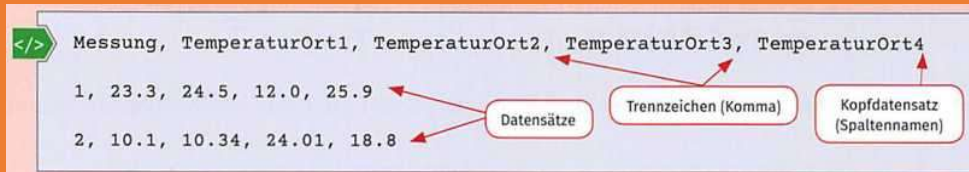
Datenformat Ein Datenformat beschreibt in welcher Struktur Daten abgelegt und wie diese bei der Verarbeitung zu interpretieren sind.
--

Allerdings muss das Datenformat bekannt sein, damit man damit arbeiten kann. Dies ist oftmals nicht der Fall und somit ist ein Datenaustausch zwischen verschiedenen Programmen nicht möglich. Deswegen wurden Datenformate entwickelt, welche sehr einfach und universell gehalten sind, um auch einen Austausch von Daten zwischen nicht kompatiblen Programmen zu ermöglichen.

Gängige Datenformate zur Datenübertragung

CSV (Comma Separated Values) - Dateiendung: .csv

Beim CSV-Format werden Textdateien zur Speicherung und zum Austausch von Daten verwendet. Die einzelnen Daten werden dabei durch einen Separator getrennt in der Datei abgelegt. In der Regel ist dieser Separator ein Komma. Es sind aber auch andere Zeichen wie Semikolon, Doppelpunkt, Tabulatorzeichen, Leerzeichen oder andere Zeichen üblich. Einen einheitlichen Standard für den Aufbau gibt es nicht. In den meisten Fällen werden die Daten in Form von Listen abgelegt. Sollen kompliziertere Datenstrukturen abgebildet werden, werden statt dem CSV häufig die Datenformate XML oder JSON eingesetzt, dennoch ist es auch mit CSV-Dateien möglich, geschachtelte Datenstrukturen abzubilden. Das CSV-Format stellt ein sehr einfaches Format dar, welches nahezu jedes Computerprogramm versteht. Somit kann auch auf einfache Weise der Datenaustausch zwischen inkompatiblen Programmen ermöglicht werden.



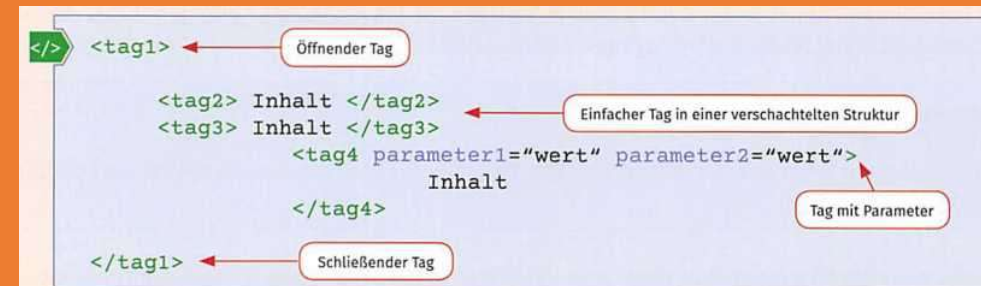
Beispiel:

```
Datei Bearbeiten Format Ansicht ?
Vorname;Nachname;PLZ;Ort;Straße;Telefon;E-Mail
Max;Mustermann;12345;Musterort;Musterstraße 1;01234/987654;max@mustermail.de
Marie;Musterfrau;23456;Musterstadt;Musterweg 2;02345/456123;marie@mustermail.de
```

XML (Extensible Markup Language) - Dateiendung: .xml

XML ist eine Auszeichnungssprache und ein Datenformat, welches u.a. zur Darstellung hierarchisch strukturierter Daten innerhalb einer Textdatei genutzt wird. Ein XML-Dokument besteht aus Inhalt und Textauszeichnungen (Markup). Auszeichnungen, die in spitze Klammern („<“ „>“) eingeschlossen sind, heißen Tags. Sie treten meist paarweise auf und schließen dann einen Inhalt ein. Tags und eingeschlossener Inhalt bilden ein Element. In XML ist es - im Gegensatz zu HTML - auch möglich, eigene Tags zu definieren.

Zusätzlich kann ein Tag noch einen oder mehrere Parameter haben. Parameter bestehen immer aus einem Namen und einem Wert. Der Wert wird mit doppeltem Anführungszeichen umschlossen und mit einem Gleichheitszeichen zugewiesen. Die Hierarchien in XML werden dadurch erzeugt, dass Tags ineinander geschachtelt werden.



Beispiele:

```
<?xml version="1.0" ?>
<Adressbuch>
  <Kunde Name="Max Mustermann">
    <Adresse>
      <Strasse>Musterstrasse 14</Strasse>
      <PLZ>12345</PLZ>
      <Ort>Musterhausen</Ort>
    </Adresse>
  </Kunde>
  <Kunde Name="August Ausgedacht">
    <Adresse>
      <Strasse>Gibts-Nicht-Strasse 37</Strasse>
      <PLZ>67890</PLZ>
      <Ort>Augsburg</Ort>
    </Adresse>
  </Kunde>
</Adressbuch>
```

```
<?xml>
<estrich>
  <kiste farbe="braun">
    <ding gewicht="800">Fotoalbum</ding>
    <ding gewicht="100">Impfpass</ding>
    <ding gewicht="1200">Rollschuhe</ding>
  </kiste>
  <kiste farbe="weiss">
    <ding gewicht="80">alte Socken</ding>
    <ding gewicht="245">leere Verpackung einer Digitalkamera</ding>
    <ding gewicht="120">Kochschürze</ding>
    <ding gewicht="65">Kochlöffel</ding>
  </kiste>
</estrich>
```

JSON (JavaScript Object Notation) - Dateiformat: Json

JSON ist ein schlankes Datenaustausch- und Textformat, welches sowohl für Menschen als auch für Programme einfach zu lesen und zu schreiben ist. Dazu verwendet es Schlüssel-Wert-Paare und eine Formatierung mit geschweiften Klammern. Die Daten können beliebig verschachtelt werden. Als Zeichenkodierung benutzt JSON standardmäßig UTF-8. Aber auch UTF-16 und UTF-32 sind möglich (UTF). JSON basiert auf JavaScript, ist aber unabhängig von der Programmiersprache einsetzbar und ist wesentlich einfacher und schlanker als XML. Deshalb verwenden viele Desktop- und Web-Anwendungen als auch mobile Apps das JSON-Format. Für die Verarbeitung von JSON wird ein geeigneter Parser benötigt. Zentrale Bestandteile sind die oben erwähnten Schlüssel-Wert-Paare und Formatierungszeichen, wie runde und geschweifte Klammern, Doppelpunkte oder das Semikolon.

```
<?xml> {
  "Breite": 800,
  "Höhe": 600,
  "Titel": "Berglandschaft",
  "Autor" {
    "Name": "Mustermann",
    "Vorname": "Max"
  },
  "Preisliste": [ 12.73, 15.22, 10.66 ],
}
```

Schlüssel-Paar-Werte

Formatierungszeichen: { } " " : ,

Verschachtelung

Array

Beispiele:

```
<?xml> [
  {
    "trackid": "AA-1234",
    "reported_dt": "12/31/2019 23:59:59",
    "longitude": -111.12500000,
    "latitude": 33.37500000
  },
  {
    "trackid": "BB-7890",
    "reported_dt": "12/31/2019 23:59:59",
    "longitude": -113.67500000,
    "latitude": 35.87500000
  },
  {
    "trackid": "CC-4545",
    "reported_dt": "12/31/2019 23:59:59",
    "longitude": -115.57500000,
    "latitude": 37.67500000
  }
]
```

```
<?xml> {
  "bands": [
    {
      "name": "Metallica",
      "description": "Metallica is an American heavy metal band. The band was formed in 1981 in Los Angeles by vocalist/guitarist James Hetfield and drummer Lars Ulrich, and has been based in San Francisco for most of its career.",
      "image": "metallica.jpg",
      "year_established": 1981,
      "albums": [
        {
          "name": "Kill 'Em All",
          "year": 1983
        },
        {
          "name": "Ride the Lightning",
          "year": 1984
        }
      ]
    }
  ]
}
```

LE 2 – Datenzugriff und -übertragung

Datenquellen können auf verschiedene Art und Weise angesprochen werden. Der Transport der Daten von einer externen Datenquelle erfolgt i.d.R. über verschiedene Netzwerkprotokolle wie FTP (File Transfer Protocol), HTTP (Hypertext Transfer Protocol), NFS (Network File Service) oder SMB (Server Message Block). Wenn Inhalte verschlüsselt übertragen werden müssen, kommen z. B. SFTP oder FTPS zum Einsatz. Auf der anderen Seite gibt es auch unzählige APIs, mit deren Hilfe Daten aus Datenquellen genutzt werden können. Sie bieten i.d.R. mehr Anpassungsmöglichkeiten und eine vielseitigere Auswahl an Zugriffsmethoden.

Da das Internet zu einer wichtigen Datenquelle und Ort des Datenaustausches geworden ist, sind die meisten Anwendungen heute in irgendeiner Form mit diesem verbunden. Für den Zugriff auf Datenquellen im Internet stehen z. B. WebDAV, SOAP oder REST zur Verfügung.

Zugriff auf Datenquellen im Web über SOAP und REST

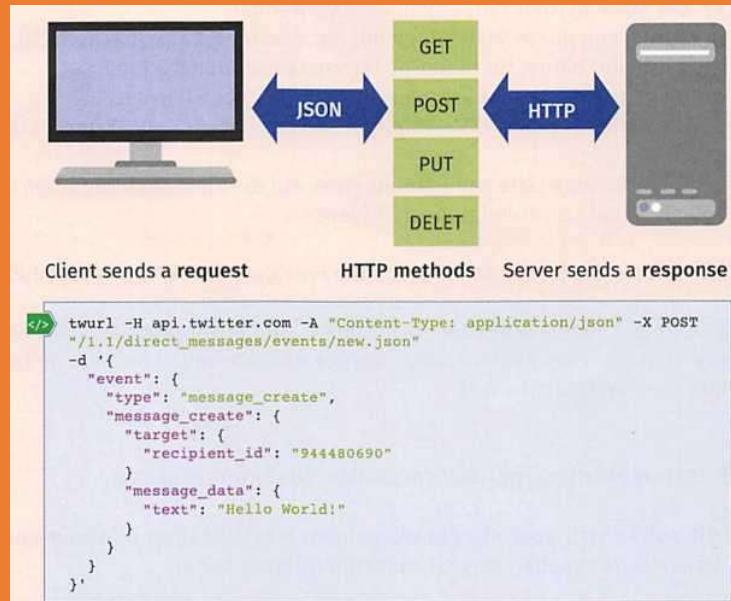
SOAP (Simple Object Access Protocol)

Bei SOAP handelt es sich um ein Netzwerkprotokoll, mit dem sich Daten austauschen lassen. Es ist ein Industriestandard des World Wide Web Consortium (W3C). SOAP ist sehr umfangreich und komplex und durch den Protokollcharakter und die damit verbundenen Regeln erhöht sich der Overhead, was u.a. zu längeren Seitenladezeiten führen kann. Auf der anderen Seite bieten die integrierten Standards, wie Sicherheit, Atomizität, Konsistenz, Isolation und Dauerhaftigkeit (ACID) (siehe auch Kapitel 3.7.1, S. 266), eine Reihe von Eigenschaften zur Gewährleistung zuverlässiger Datenbanktransaktionen. Der Austausch der Daten erfolgt bei SOAP über Nachrichten, welche auf einem XML-Information-Set basieren. Wenn eine Datenanforderung an eine SOAP-API gesendet wird, kann sie über eines der Protokolle der Anwendungsschicht verarbeitet werden: HTTP (für Webbrowser), SMTP (für E-Mail), TCP und andere. Sobald eine Anforderung empfangen wird, müssen SOAP-Nachrichten als XML-Dokumente zurückgegeben werden. Das Erstellen des entsprechenden XML-Codes kann unter Umständen sehr komplex werden. Deshalb bieten manche Programmiersprachen sogenannte Metasprachen an, welche den Aufwand der Erstellung von XML-Dokumenten reduzieren. Eine solche Metasprache ist z. B. WDSL (Web Services Description Language). WDSL ist eine plattform-, Programmiersprachen- und protokollunabhängige Beschreibungssprache für Netzwerkdienste (Webservices) und dient zum Austausch von Nachrichten auf XML-Basis. Mithilfe von WDSL lassen sich in SOAP Schnittstellen von Web-Services beschreiben (Funktionsweise von Diensten, Definitionen von Nachrichtenformaten usw.), ohne dass direkt XML-Dokumente erstellt werden müssen.

```
</> <?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="..."
  SOAP-ENV:encodingStyle="...">
  <SOAP-ENV:Header>
    ...
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    ...
    <SOAP-ENV:Fault>
      ...
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

REST (Representational State Transfer)

REST ist eine moderne Alternative zu SOAP. Es kommt mit weniger Ressourcen aus und lässt sich einfacher erlernen bzw. anwenden. Cloud-Anbieter wie AWS, VMware, Azure und andere setzen heute fast ausschließlich auf REST. Neben dem Zugriff auf Datenquellen im Web ermöglicht die REST API auch eine Kommunikation von Maschine zu Maschine. Die REST-API nutzt HTTP-Anfragen, um per PUT, GET, POST und DELETE auf Daten zuzugreifen. Bei der REST-Schnittstelle ist nicht genau festgelegt, wie die Services implementiert werden müssen.

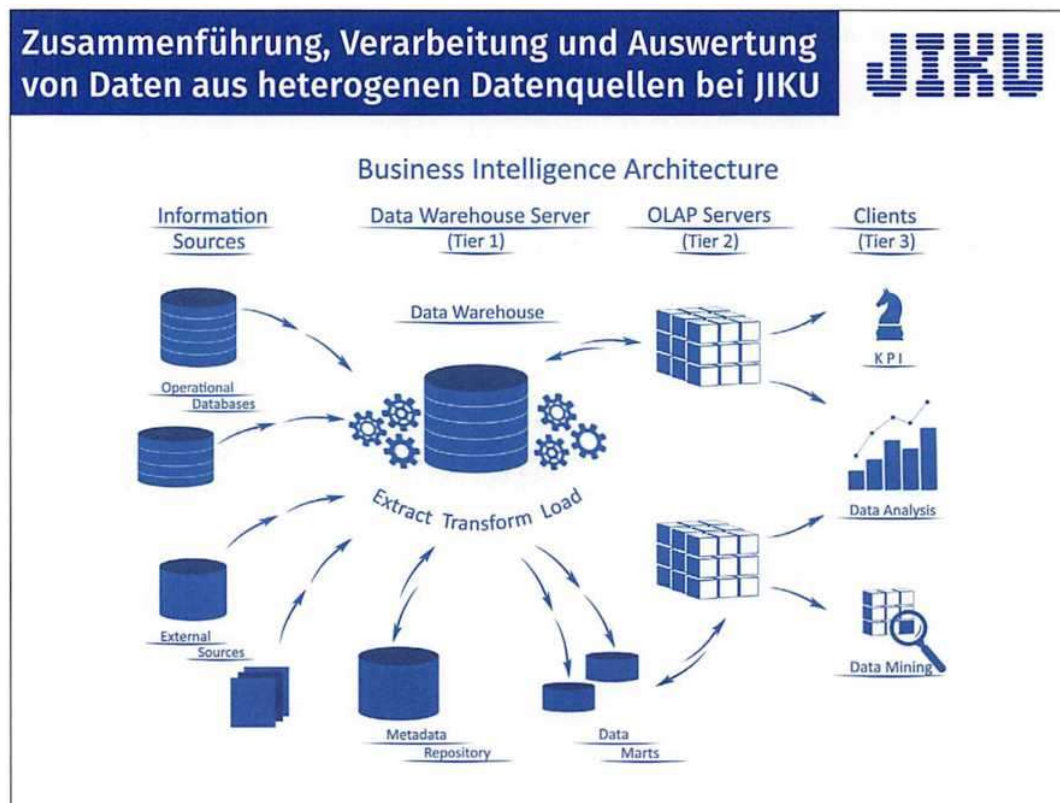


Es gibt jedoch sechs verschiedene Prinzipien („Constraints“), die ein Service-Dienst aufweisen sollte:

- Client-Server-Modell
- Zustandslosigkeit
- Caching
- Einheitliche Schnittstelle
- Layered System
- Code-On-Demand

Stellt sich die Frage, wann SOAP und wann REST angewendet werden sollte. SOAP-Webservices bieten integrierte Sicherheit und Transaktions-Compliance, was vielen Unternehmensanforderungen entspricht, sind aber aufwendiger zu erstellen. SOAP ist in den Augen vieler Entwickler komplex und schwer zu handhaben. REST-API sind ideal für moderne Anwendungen geeignet, z. B. IoT oder mobile Anwendungen. REST kann allein mit den vier HTTP-„Operationen“ GET, POST, PUT und DELETE fast alle Aufgaben ausführen und muss zum Bereitstellen der Antwort kein XML verwenden. Damit ist REST zukunftsorientierter und viele öffentliche Schnittstellen verfolgen diesen Ansatz. Da SOAP aber älter ist als REST, nutzen heutzutage noch viele Altsysteme SOAP. Insofern wird man SOAP auch in Zukunft nicht umgehen können.

LE 3 - Daten heterogener Datenquellen zusammenführen



Unternehmen beziehen ihre Daten aus einer wachsenden Anzahl von teilweise sehr unterschiedlichen Datenquellen. Diese können sowohl externe als auch interne Datenquellen sein. In vielen Fällen enthalten diese Quellen gleiche Daten, welche jedoch teilweise anders beschrieben und gespeichert sind. So wird beispielsweise in einem System eine Ortsangabe als „Ort“ und in dem anderen als „Location“ bezeichnet oder zu den Kerndaten wie Adresse und Name kommen auch noch andere Zusatzinformationen dazu. In all diesen Fällen spricht man von heterogenen Datenquellen.

Heterogen bedeutet „uneinheitlich“, „andersartig“ oder, dass etwas aus Ungleichartigem zusammengesetzt ist. Das Gegenteil von heterogen ist homogen.

Es gibt verschiedene Formen, durch welche die Heterogenität von Datenquellen gekennzeichnet sein kann. Umso mehr Punkte hiervon zutreffen, umso mehr unterscheiden sich diese voneinander.

Formen von Heterogenität		
Form	Beschreibung	Beispiel
Technisch	Von technischer Heterogenität spricht man, wenn sich die Zugriffsschnittstelle unterscheidet.	XPath, SQL usw.
Syntaktisch	Syntaktische Heterogenität liegt vor, wenn Daten z. B. in unterschiedlichen Formaten gespeichert werden.	20.03.2021, 2021-03-20, 20. März 21 usw.

Formen von Heterogenität		
Form	Beschreibung	Beispiel
Zugrunde liegendes Datenmodell	Heterogenität bezüglich des Datenmodells liegt vor, wenn die verwendeten Datenmodelle der Quellen verschieden sind.	dokumentenbasiert, relational usw.
Strukturell	Von struktureller Heterogenität spricht man, wenn sich die Struktur der Speicherung der Daten unterscheidet. Das Datenmodell ist aber gleich.	In der einen Quelle werden z. B. die Adressdaten direkt in der Kundentabelle verwaltet und in einer anderen Quelle sind sie in einer extra Tabelle ausgelagert und mit der Kundentabelle verknüpft.
Semantisch	Semantische Heterogenität liegt vor, wenn sich die Bedeutung, Interpretation und Art der Nutzung des Datenmodells unterscheidet.	In einer Quelle wird die Ortsangabe als „Ort“ und in einer anderen als „Location“ bezeichnet.

In den Unternehmen besteht nun das Ziel, die Daten vollständig und effizient aus verschiedenen Datenquellen zu einer strukturierten Einheit zusammenzuführen, um effektiv auf diese zugreifen zu können. Man spricht in diesem Zusammenhang auch von Informationsintegration.

Informationsintegration

Unter Informationsintegration versteht man das Zusammenführen von Daten aus verschiedenen Datenquellen mit i.d.R. heterogenen Datenstrukturen in eine gemeinsame einheitliche Datenstruktur.

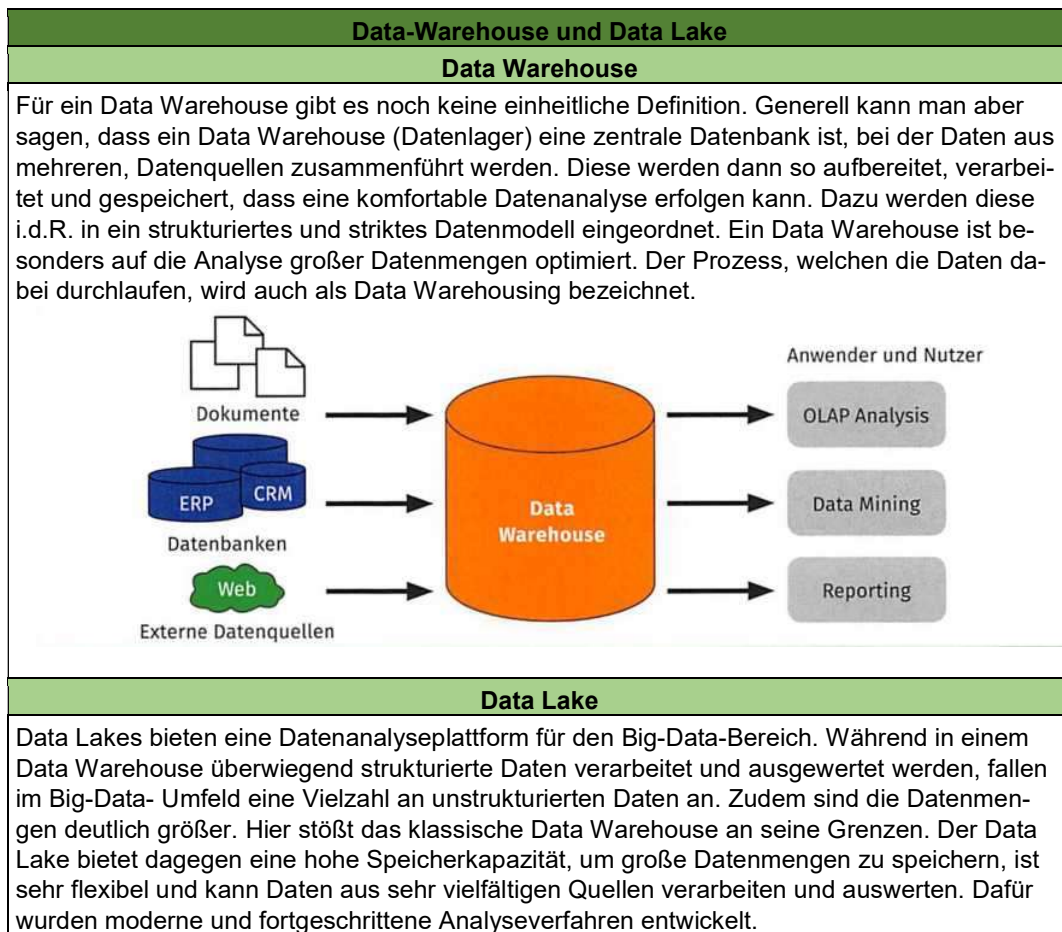
Dabei kommt es vor, dass die Daten der verschiedenen Quellen teilweise redundant sind. Diese Redundanz muss bei der Zusammenführung der Daten beseitigt werden. Redundante Daten bieten aber den Vorteil, dass sie sich gut zur Verifikation nutzen lassen.

Möglichkeiten der Integration von Daten aus unterschiedlichen Datenquellen

Materialisierte oder physische Integration	Hier werden die Daten aus verschiedenen Datenquellen in eine zentrale Datenbasis überführt, welche eine einheitliche Struktur für alle Daten bereitstellt. Dieses Prinzip findet beispielsweise in Data Warehouses oder Data Lakes Anwendung. Vorteile: • Durch die zentrale Haltung der Daten kann das System entsprechende Daten relativ schnell zur Verfügung stellen. • Es wird i. d. R. eine höhere Qualität der Daten sichergestellt als bei einer virtuellen Lösung. Nachteile: • Daten sind in vielen Fällen nicht immer aktuell. • Das Hinzufügen, Ändern oder Löschen von Datenquellen kann einen nicht zu unterschätzenden Aufwand nach sich ziehen. • Es muss i.d.R. entsprechende Hardware beschafft und gewartet werden.
---	--

Virtuelle oder logische Integration	Hier verbleiben die Daten in den verschiedenen Datenquellen und die Integration findet erst bei einer Anfrage statt. Dieses Prinzip findet beispielsweise in föderierten Datenbanksystemen oder mediatorbasierten Informationssystemen Anwendung. Vorteile: • Die Daten befinden sich immer auf aktuellem Stand. • Das Hinzufügen, Ändern oder Entfernen von Datenquellen kann sehr flexibel gehandhabt werden. • Nachteile: • Die Abfrage der Daten kann mitunter sehr lange dauern, da diese immer aus unterschiedlichen Quellen zusammengeführt werden müssen. • Die Qualität der Daten ist i. d. R. schlechter als bei einer physischen Lösung.
--	--

Wie erwähnt findet die materialisierte oder physische Integration beim Data-Warehouse und beim Data Lake statt.



Die Verwendung von Data Warehouses und Data Lakes, um Daten aus heterogenen Datenquellen zu nutzen, ist teilweise sehr aufwendig und zeitintensiv. Andere Nachteile sind, dass sie nur in gewissen Intervallen aktualisiert werden und damit unter Umständen nicht mehr synchron mit den Originaldaten sind. In diesen Fällen schafft die oben erwähnte Datenvirtualisierung Abhilfe. Da hierbei die Daten an ihren ursprünglichen Speicherorten verbleiben, sind diese auch immer aktuell. Dadurch können die Ergebnisse der Datenanalyse und -auswertung teilweise erheblich verbessert werden.

Programmierparadigmen				
Imperative Programmierparadigmen			Deklarative Programmierparadigmen	
Strukturierte Programmierung	Prozedurale Programmierung	Objektorientierte Programmierung	Funktionale Programmierung	Logische Programmierung

Die einzelnen Programmierparadigmen werden nun etwas genauer betrachtet. Weil das objektorientierte Programmierparadigma aktuell in vielen Bereichen der Softwareentwicklung als „Standard“ angesehen wird, wird darauf vertiefend am Tag 5 eingegangen.

LE 2 - Imperative Programmierparadigmen

Bei imperativen Programmierparadigmen wird im Programm mithilfe von Befehlen präzise beschrieben, welche Schritte in welcher Reihenfolge abzuarbeiten sind. Dabei werden verschiedene Kontrollstrukturen, z. B. Verzweigungen und Schleifen, verwendet. Die ersten Computerprogramme bauen auf diesem Ansatz auf. Der daraus resultierende Programmcode ist deshalb leicht verständlich, aber auch sehr umfangreich.

```

367 int lcd_create_map_value_to_empty
368 {
369     memset(empty, 0, 8);
370     int i = 0;
371     int tmp;
372
373     tmp = percent1 / 10;
374     printf("percent1 = %d, tmp = %d\n", percent1, tmp);
375     for(i = 7; i >= 0; i--)
376     {

```

Imperatives Programmierparadigma

Beim imperativen Programmierparadigma wird im Programm beschrieben, wie ein Problem zu lösen ist. Die Lösung wird als eine Folge von einzelnen Anweisungen beschrieben.

Die wichtigsten imperativen Programmierparadigmen sind das strukturierte, das prozedurale und das objektorientierte Programmierparadigma.

Imperative Programmierparadigmen

Strukturiertes Programmierparadigma

Bei dem strukturierten Programmierparadigma handelt es sich um eine Form der Programmierung, die im Zuge der sogenannten Softwarekrise Mitte der 1960er-Jahre entstanden ist und eine Erweiterung des ursprünglichen imperativen Programmierparadigmas darstellt. Der Hauptansatz besteht darin, dass auf bestimmte Sprunganweisungen (Goto) verzichtet wird und stattdessen Kontrollstrukturen wie Verzweigungen und Schleifen zum Einsatz kommen. Das strukturierte Programmierparadigma beinhaltet einen ersten Ansatz in Richtung des prozeduralen Programmierparadigmas, indem es eine Zerlegung des Programms in Teilprogramme oder Funktionen verlangt. Programmiersprachen, welche den strukturierten Ansatz verfolgen, sind u.a. C oder Pascal.



Beispiel: Programm in Pascal

```

program Beispiel;
var a : integer;
begin
    a := 10;
    if a = 10 then
        writeln ('a ist gleich 10');
    end.

```

Prozedurales Programmierparadigma

Das prozedurale Programmierparadigma baut auf dem strukturierten Programmierparadigma auf. Es zielt darauf ab, Quelltexte wiederverwendbar zu machen. Dazu wird das Programm in überschaubare Teile aufgeteilt. Diese Teile werden als Prozeduren, Unterprogramme oder

Funktionen bezeichnet. Dadurch soll der Programmcode übersichtlicher und Code-Wiederholungen (Redundanzen) vermieden werden. Diese Vorgehensweise bietet den Vorteil, dass man universell einsetzbare Prozeduren entwickeln kann. Programmiersprachen, welchen den prozeduralen Ansatz verfolgen, sind u.a. C, FORTRAN, COBOL, Pascal.



Beispiel: Programm in C

```
#include<stdio.h>
int addiere(int summand1, int summand2) {
    return (summand1 + summand2);
}
int main() {
    int summe = addiere(3,7);
    printf("Summe von 3 und 7 ist %d\n", summe);
    return 0;
}
```

Objektorientiertes Programmierparadigma

Das objektorientierte Programmierparadigma stellt eine Weiterentwicklung der anderen Paradigmen dar und ist heutzutage der Programmierstandard in vielen Bereichen der Softwareentwicklung. Deswegen wird dieses Programmierparadigma im nächsten Abschnitt nochmals vertiefend erläutert.

LE 3 - Deklarative Programmierparadigmen

Bei deklarativen Programmierparadigmen wird im Vergleich zu imperativen Programmierparadigmen nur das gewünschte Ergebnis, also das, was die Software leisten soll, im Programm beschrieben. Die einzelnen Schritte, wie die Software zu diesem Ergebnis gelangen soll, werden dagegen nicht programmiert. Dies führt dazu, dass der Quelltext deklarativer Programme aufgrund des hohen Abstraktionsgrades schwerer verständlich ist. Dafür ist er aber auch sehr kurz und präzise. Das hohe Abstraktionsniveau der deklarativen Programmierung hat des Weiteren den Vorteil, dass die Ausführung des Programms durch Algorithmen klar von der eigentlichen Programmentwicklung getrennt ist. Dadurch lässt sich die Wartung und Optimierung des Systems unabhängig von den eigentlichen Anwendungen durchführen.



Deklaratives Programmierparadigma

Beim deklarativen Programmierparadigma wird im Programm beschrieben, was das Problem ist. Wie dieses Problem gelöst wird, wird dem System überlassen.

Die zwei wichtigsten Arten der deklarativen Programmierung sind das funktionale und das logische Programmierparadigma.

Deklarative Programmierparadigmen

Funktionales Programmierparadigma

Bei der funktionalen Programmierung besteht das Programm nur aus einer Reihe von Funktionsaufrufen. Reine

Wertzuweisungen wie bei der imperativen Programmierung gibt es hier nicht. Das funktionale Programmierparadigma wird bei der Erstellung von Programmen für die künstliche Intelligenz oder im Compilerbau eingesetzt. Funktionale Programmiersprachen beruhen i.d.R. auf dem sogenannten Lambda-Kalkül. Beispiele funktionaler Programmiersprachen sind Lisp, Haskell oder F#.



Beispiel: Quicksort in Haskell

```
qsort [] = []
qsort (p:xs) = qsort kleiner ++ [p] ++ qsort groesser
  where groesser = filter (< p) xs
        kleiner = filter (>= p) xs
```

Logisches Programmierparadigma

Das logische Programmierparadigma beruht auf der mathematischen Logik. Das Programm selbst ist eine Datenbasis, die aus Fakten und Regeln

besteht. Jegliche Anfragen an das Programm werden verarbeitet, indem der Interpreter auf diese Fakten und Regeln zurückgreift und sie anwendet, um zum gewünschten Ergebnis zu kommen. Ein Beispiel für logische Programmiersprachen ist die Sprache Prolog.



Beispiel: Regeln in Prolog

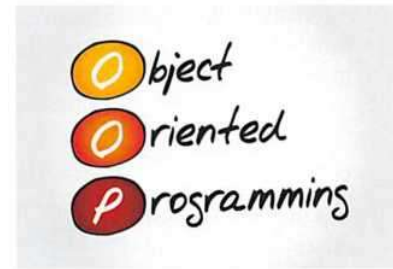
```
kind(K,E) :- elternteil(E,K).
mutter(M,K) :- weiblich(M), elternteil(M,K).
```

Die deklarative Programmierung basiert auf einem für den Menschen unüblichen Denkmodell. Es wird ein Lösungszustand beschrieben, aber nicht der Weg dahin. Menschen sind es aber gewohnt, in Prozessen auf ein Ziel hin zu denken, was die Entwicklung solcher Programme erst einmal erschwert. In der Praxis kommen heute meist Mischformen der Paradigmen zum Einsatz, d.h. deklarative Programmiersprachen werden um imperative Methoden und Konzepte ergänzt. Das erhöht aber wiederum die Fehleranfälligkeit der Programme und es kann die Lesbarkeit des Quelltextes beeinträchtigen.

TAG 5 - Das objektorientierte Programmierparadigma beschreiben

Die objektorientierte Programmierung baut auf den Verfahren der strukturierten und der prozeduralen Programmierung auf und verfolgt die Grundidee, eine Software so zu entwickeln, dass der Aufbau des Programms sich an den Strukturen (Objekten) der Wirklichkeit ausrichtet. Dazu werden in der Analysephase Informationen über die auftretenden Objekte des entsprechenden Problembereichs gesammelt und diese danach abstrakt beschrieben. Die Umsetzung dieses Programmierstils erfordert die Einführung und Umsetzung verschiedener Konzepte, z.B.

das Konzept der Klasse, der Vererbung oder des dynamischen Bindens. Dabei sind aber alle diese Konzepte nicht starr vorgegeben und unterliegen einer fortlaufenden Entwicklung. Zudem werden nicht alle Konzepte von allen objektorientierten Programmiersprachen unterstützt. Exemplarisch steht hierfür das Konzept der Mehrfachvererbung, welches in vielen objektorientierten Programmiersprachen gar nicht oder nur teilweise umgesetzt wird. Durch die OOP soll erreicht werden, dass die Qualitätsstandards der ISO/IEC 9126, z. B. Zuverlässigkeit oder Änderbarkeit, bei der Entwicklung einer Software eingehalten werden. Weiterhin wird durch die OOP auch die Flexibilität und Wiederverwendbarkeit von Programmen und Programmteilen gefördert, was wiederum eine schnellere Entwicklung von neuen Programmen ermöglicht.



LE 1 - Abstraktion

Wie schon erwähnt, versucht man in der objektorientierten Programmierung, Objekte der realen Welt im Programm abzubilden. Zur besseren Verwaltung gleichartiger Objekte wird in der OOP das Konzept der Klasse angewandt.

Objekt

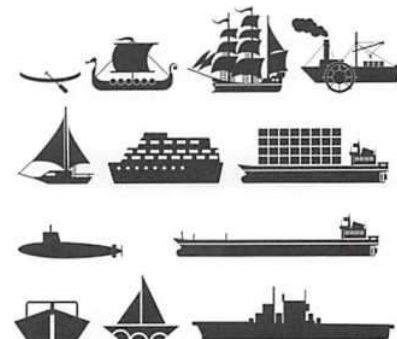
Ein Objekt ist ein abstrakter oder realer Gegenstand, welcher durch seinen Zustand (Eigenschaften mit konkreten Werten) und sein Verhalten (eine Anzahl bestimmter Methoden) beschrieben wird.

Dabei wird versucht, die Eigenschaften und Methoden gleichartiger Objekte zu abstrahieren und für diese Objekte jeweils eine sogenannte Klasse zu generieren. Man kann auch sagen, dass eine Klasse die gemeinsamen Eigenschaften und Methoden einer Menge von gleichartigen Objekten beschreibt.

Klasse

Eine Klasse ist eine Beschreibung (Vorlage, Bauplan) für eine Menge gleichartiger Objekte. Ein Exemplar (Instanz) einer Klasse ist wiederum ein konkretes Objekt.

Soll beispielsweise ein Computerspiel geschrieben werden, in dem Wasserfahrzeuge eine Rolle spielen, müssen zunächst einmal alle Wasserfahrzeuge und ihr Kontext im zukünftigen Spiel erfasst werden. Dabei könnte es, wie nebenstehend dargestellt, sehr unterschiedliche Wasserfahrzeuge geben. So unterschiedlich diese Wasserfahrzeuge auch sind, sie besitzen gemeinsame Eigenschaften (Attribute) und Methoden, welche durch eine Klasse beschrieben werden können. Gemeinsame Eigenschaften könnten beispielsweise der Name, der Tiefgang, aber auch das Baujahr oder der Verkaufspreis sein. Die Eigenschaften, welche dabei eine Rolle spielen, hängen ganz von dem entsprechenden Kontext im Spiel ab. Das Gleiche gilt für die Methoden, also die



Dinge, welche alle Wasserfahrzeuge tun können. Ein Beispiel für eine solche Methode wäre schwimmen(). Die Klasse selbst könnte den Namen Wasserfahrzeug bekommen. Um die Eigenschaften und Methoden einer Klasse übersichtlich darzustellen, wird diese im Allgemeinen durch ein UML-Klassendiagramm beschrieben. Nebenstehend soll exemplarisch aufgezeigt werden, wie eine solche Klasse „Wasserfahrzeug“ als Klassendiagramm aussehen könnte. Aus dieser Klasse lassen sich dann im Programm Objekte (Wasserfahrzeuge) bilden. Jedes Objekt erhält nun für seine Eigenschaften konkrete Werte. In der Tabelle sind einige Beispiele für Objekte mit den entsprechenden Werten aufgeführt.



UML-Klassendiagramm

Objekt	Abbildung	Eigenschaften mit konkreten Werten
wasserfahrzeug1		name: Airwave tiefgang: 150 baujahr: 1930 besitzer: Maier
wasserfahrzeug2		name: Meeresluft tiefgang: 125 baujahr: 1980 besitzer: Müller
wasserfahrzeug3		name: Royal tiefgang: 115 baujahr: 2020 besitzer: Schulze

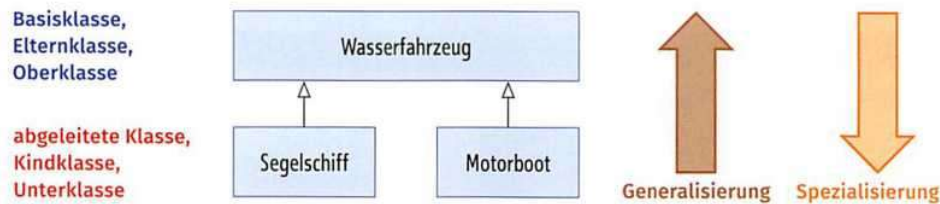
LE 2 - Kapselung

Das Konzept der Kapselung beinhaltet das Verbergen von Implementierungsdetails eines Objektes. Auf die Eigenschaften (Daten) eines Objektes kann nicht mehr direkt zugegriffen werden, sondern nur über definierte Schnittstellen. Auf diese Weise soll sichergestellt werden, dass die Daten nicht in unerwarteter Weise gelesen oder geändert werden. Die Methoden eines Objektes bilden diese Schnittstellen und der Zugriff auf die einzelnen Eigenschaften und Methoden wird durch Zugriffsmodifizier geregelt. Zugriffsmodifizier werden in der Klassenbeschreibung von Objekten allen Komponenten (Eigenschaften und Methoden) zugeordnet. Durch den bewussten Einsatz von diesen wird eine Kapselung der Daten erreicht, sodass sie von außen nicht mehr direkt manipulierbar sind.

Zugriffsmodifizier	Beschreibung
public	Auf öffentliche (public) Komponenten darf uneingeschränkt zugegriffen werden.
protected	Auf geschützte (protected) Komponenten kann nur innerhalb der eigenen Klasse und aus Klassen, die von dieser Klasse abgeleitet wurden, zugegriffen werden.
private	Ein Zugriff auf private Komponenten ist nur innerhalb der Klasse erlaubt.

LE 3 - Vererbung

Neben der Kapselung ist die Vererbung ein weiteres grundlegendes Konzept der OOP. Über die Vererbung kann man eine neue Klasse auf Basis einer bereits vorhandenen Klasse (Basisklasse, Elternklasse) erzeugen. Die abgeleitete Klasse übernimmt (erbt) dabei sämtliche Eigenschaften und Methoden der Basisklasse und es werden neue Eigenschaften und Methoden hinzugefügt. Die abgeleitete Klasse erweitert somit die Basisklasse.



Generell wird bei der Vererbung noch zwischen Einfachvererbung und Mehrfachvererbung unterschieden. Dabei bezieht sich die Einordnung immer nur auf die vorhergehende Vererbungsstufe.

Arten von Vererbung		
Art	Beschreibung	Beispiel
Einfachvererbung	Bei der Einfachvererbung hat eine abgeleitete Klasse genau eine Basisklasse, von der sie erbt.	
Mehrfachvererbung	Bei der Mehrfachvererbung hat eine abgeleitete Klasse mindestens zwei Basisklassen, von denen sie erbt.	

In dem Beispiel der Einfachvererbung werden aus der Klasse „Wasserfahrzeug“ die neuen Klassen „Segelschiff“ und „Motorboot“ abgeleitet. Dabei erben diese beiden Klassen alle Eigenschaften und Methoden vom Wasserfahrzeug, welche *public* oder *protected* sind. Sie können um sinnvolle neue Eigenschaften und Methoden erweitert werden, welche nur für diese speziellen Klassen gelten. So könnte die Klasse „Segelschiff“ beispielsweise die Eigenschaft „anzahl-Masten“ erhalten und die Klasse „Motorboot“ die

Eigenschaft „anzahlMotoren“. Bei der Mehrfachvererbung erbt die Klasse „Wasserflugzeug“ gleich die Eigenschaften und Methoden von zwei Klassen, z. B. die Methoden „fliegen()“ und „schwimmen()“. Da es aber bei der Mehrfachvererbung zu Problemen kommen kann, wird dieses Konzept nur in wenigen objektorientierten Programmiersprachen direkt umgesetzt.

LE 4 - Polymorphie

Neben der Abstraktion, Kapselung und Vererbung ist Polymorphie ein weiteres Konzept der OOP. Das Wort selbst kommt aus dem Griechischen und bedeutet Vielgestaltigkeit. In der Programmierpraxis bedeutet es, dass verschiedene Objekte bei Aufruf derselben Methode ein unterschiedliches Verhalten zeigen können. Durch dieses Prinzip ist es möglich, dass einer Referenz eines Objekttyps einer Basisklasse nicht nur Objekte der Basisklasse, sondern auch aller abgeleiteten Klassen zugewiesen werden können. Es wird erst zur Laufzeit festgestellt, welche Methoden zu welchem Objekt gehören. Dadurch ist es möglich, in einem gewissen Rahmen flexibel auf sich ändernde Anforderungen zu reagieren.

Arten von Polymorphie in der OOP	
Dynamische Bindung	Da einer Referenz eines Objekttyps einer Basisklasse nicht nur Objekte der Basisklasse, sondern auch aller abgeleiteten Klassen zugewiesen werden können, steht in vielen Fällen zur Zeit der Übersetzung des Programms noch nicht fest, ob die aufgerufenen Methoden einer Basisklasse oder einer abgeleiteten Klasse zuzuordnen sind. Dieses wird erst zur Laufzeit entschieden. Dieser Mechanismus wird als dynamisches Binden oder auch spätes Binden bezeichnet. Durch das Konzept der dynamischen Bindung können die Möglichkeiten der Vererbung erst voll ausgeschöpft werden.
Überladen von Methoden	Das Überladen von Methoden bedeutet, dass es mehrere Methoden mit demselben Methodennamen in einer Klasse geben darf. Die Methoden müssen aber trotzdem zu unterscheiden sein. Dies kann auf zwei Arten geschehen: Der Datentyp mindestens eines Übergabeparameters ist anders als in den übrigen gleichnamigen Methoden. Die Anzahl der Übergabeparameter ist unterschiedlich. Welche Methode genau aufgerufen wird, wird dann bei jedem Methodenaufruf anhand der Parameter und ihrer Datentypen automatisch bestimmt.
Überschreiben von Methoden	Das Überschreiben von Methoden bedeutet, dass eine Methode einer Basisklasse in der abgeleiteten Klasse neu implementiert (überschrieben) wird. Damit eine Methode überschrieben werden kann, müssen folgende Bedingungen erfüllt sein: • Die Methoden müssen namentlich übereinstimmen. Die Methoden müssen im Datentyp und in der Anzahl ihrer Übergabeparameter übereinstimmen. Der Rückgabedatentyp muss übereinstimmen. Die überschreibende Methode darf in ihren Zugriffsrechten nicht mehr beschränkt sein als die überschriebene Methode. Die Zugriffsrechte dürfen jedoch gelockert werden (z. B. von <i>protected</i> auf <i>public</i>). Die zu überschreibende Methode muss auch tatsächlich vererbt werden. Privat gesetzte Methoden werden nicht vererbt und können damit auch nicht überschrieben werden.

LE 5 - Objektorientierte Programmiersprachen

Als eine objektorientierte Programmiersprache wird eine Programmiersprache bezeichnet, welche die objektorientierten Konzepte ganz oder zum größten Teil unterstützt. Dabei gibt es einerseits reine objektorientierte Sprachen, welche einen objektorientierten Programmierstil erfordern, z. B. Java. Andererseits gehören dazu Programmiersprachen, welche um OOP-Konzepte erweitert wurden, mit denen es aber theoretisch möglich ist, rein prozedurale Programme zu schreiben, z. B. ist hier C++ zu nennen. In diesem Fall wurde die Sprache C um objektorientierte Elemente erweitert. Die ältesten objektorientierten Sprachen sind Simula (1967) und Smalltalk (1970). Weitere moderne und populäre Programmiersprachen, welche den objektorientierten Ansatz unterstützen, sind in der Tabelle aufgeführt.

Übersicht über einige wichtige OOP-Sprachen

C++(1985)

C++ ist eine Programmiersprache, welche aus der Sprache C entstanden ist. Bjarne Stroustrup erweiterte C um objektorientierte Elemente, welche er zunächst „C with Classes“ nannte. Später wurde daraus C++. C++ unterstützt viele der objektorientierten Konzepte, u.a. auch die Mehrfachvererbung. Es ist aber weiterhin möglich, in C++ auch nur rein prozedurale Programme zu schreiben.

Java (1995)

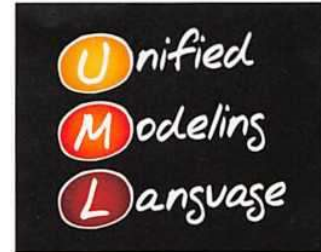
Java ist eine universell einsetzbare Programmiersprache, die rein objektorientiert ist und von der Firma Sun Microsystems entwickelt wurde. 2010 wurde Sun Microsystems von Oracle aufgekauft, welche seitdem Java weiterentwickelt. Java unterstützt die meisten objektorientierten Konzepte. Mehrfachvererbung kann aber nur indirekt mithilfe von Interfaces realisiert werden.

C# (2000)

C# (C-Sharp) ist auch eine rein objektorientierte Sprache und wurde von Microsoft entwickelt. In ihr wurden u.a. Konzepte aus den Sprachen Java, C++ und Delphi aufgegriffen und verfeinert. Die Sprache unterstützt die meisten objektorientierten Konzepte. Auch hier kann die Mehrfachvererbung nur indirekt mithilfe von Interfaces realisiert werden.

TAG 6 - Die Modellierungssprache UML verwenden

Größere Softwareprojekte können nicht ohne Vorbereitung und Planung erfolgreich erstellt werden. Abstimmungsgespräche, sowohl zwischen den einzelnen Projektmitgliedern als auch mit dem Kunden, sind notwendig. In der Regel werden für die Kommunikation und Dokumentation Modellierungssprachen, z.B. das Struktogramm oder das ERM verwendet. Durch diese wird die Kommunikation vereinfacht und Sachverhalte können übersichtlich dargestellt werden. Heutzutage wird dafür immer häufiger auf die Modellierungssprache UML (Unified Modeling Language) zurückgegriffen.



Unified Modeling Language (UML)

Unified Modeling Language (UML) ist eine Modellierungssprache zur Spezifikation, Visualisierung, Konstruktion und Dokumentation von Modellen für Softwaresysteme. Sie kann aber auch für andere Themen, z. B. zur Darstellung von Geschäftsmodellen, genutzt werden.

UML bietet ein ganzes Spektrum an Diagrammtypen für die Visualisierung der unterschiedlichsten Sachverhalte und ist besonders auf den objektorientierten Ansatz (siehe auch objektorientiertes Programmierparadigma) ausgerichtet. UML bietet Softwareentwicklern die Möglichkeit, den Entwurf auf einheitlicher Basis zu dokumentieren und zu diskutieren und wird seit 1998 als Standard angesehen. Die Verantwortung für die Standardisierung der Symbole dieser Sprache liegt bei der Object Management Group (OMG).

Geschichtliche Entwicklung von UML

Die Notwendigkeit der Planung von Softwaresystemen wurde schon in den 1970er-Jahren im Zuge der sogenannten „Softwarekrise“ erkannt. Im Laufe der Zeit wurden für die Modellierung von Softwaresystemen die unterschiedlichsten Modellierungssprachen entworfen, was dazu führte, dass sich am Ende viele inkompatible und widersprüchliche Notationen gegenüberstanden. Grady Boch, Ivar Jacobsen und Jim Rumbaugh begannen Anfang der 1990er-Jahre diese Notationen zu vereinheitlichen. Ihre Bemühungen endeten schließlich in einer Modellierungssprache mit dem Namen UML. Viele führende Unternehmen der Computerbranche wie Microsoft, Oracle und Hewlett-Packard erkannten den Vorteil einer einheitlichen Modellierungssprache und unterstützten diese Entwicklung. 1997 wurde die erste UML-Version herausgebracht. Seitdem hat sich UML als Standard durchgesetzt und wird ständig weiterentwickelt. Seit Ende der 1990er-Jahre wurde intensiv an der UML-Version 2.0 gearbeitet, welche im Jahr 2005 fertiggestellt wurde. Die aktuelle UML-Spezifikation 2.5.1 der OMG umfasst fast 800 Seiten (in Englisch) und wurde 2017 veröffentlicht.

Die aktuelle Version 2.5.1 von UML weist 14 verschiedene Diagrammtypen auf, welche in verschiedene Kategorien unterteilt werden. Allerdings haben die einzelnen Diagrammtypen teilweise mehr Gemeinsamkeiten, als es die weiter unten dargestellte Einteilung vermuten lässt. Es ist nicht verboten und sogar üblich, Elemente des einen Diagrammtyps in einem anderen Diagrammtyp zu verwenden. Das gilt insbesondere für Diagramme aus den beiden großen Kategorien untereinander.

Für die Darstellung der einzelnen Diagramme werden i.d.R. spezielle Programme verwendet. Diese finden vor allem bei der Erstellung von verschiedenen Dokumentationsunterlagen Anwendung. Bei Projektbesprechungen ist es aber vielfach üblich, die Diagramme schnell per Hand zu entwerfen und zu diskutieren. Diese werden dann später am Computer in die richtige Form gebracht. Damit diese am Computer entworfenen Modelle auch von verschiedenen Programmen verarbeitet werden können, hat die OMG ein standardisiertes Austauschformat entwickelt. Dieses Format basiert auf XML und hat den Namen XML Metadata Interchange (XMI). Neben UML lassen sich mit diesem Austauschformat auch beliebige andere Metadaten austauschen.

UML-Diagramme

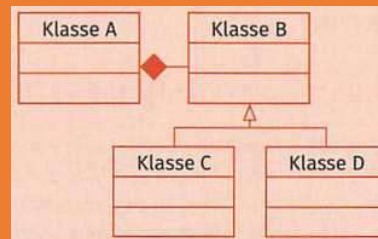
Verhaltensdiagramme		Strukturdiagramm	
Anwendungsfalldiagramm	Sequenzdiagramm	Klassendiagramm	Objektdiagramm
Aktivitätsdiagramm	Zeitverlaufdiagramm	Komponentendiagramm	Verteilungsdiagramm
Zustandsdiagramm	Kommunikationsdiagramm	Paketdiagramm	Profilidiagramm
Interaktionsübersichtdiagramm		Kompositionsstrukturdiagramm	

Die einzelnen Diagramme werden in Strukturdiagramme oder Verhaltensdiagramme eingeteilt. Strukturdiagramme beschreiben im Allgemeinen den Zustand eines Systems und sind somit statisch. Sie stellen den Aufbau und die Beziehungen der Komponenten zu einem bestimmten Zeitpunkt dar. Verhaltensdiagramme dagegen bilden die Aktivitäten und Abläufe in diesem System ab, d.h., sie sind dynamisch und orientieren sich am Zeitablauf.

Übersicht UML-Diagramme

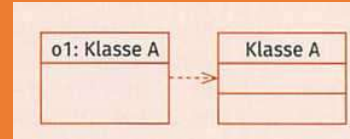
Klassendiagramm (Class Diagram)

Das Klassendiagramm ist der wichtigste Diagrammtyp der UML. Mit diesem Diagramm werden Klassen beschrieben und die Beziehungen zwischen diesen dargestellt. Es stellt einen Bauplan für die Objekte des Programms dar.



Objektdiagramm (Object Diagram)

Das Objektdiagramm ist eine Spezifizierung des Klassendiagramms. Es stellt die Beziehungen der tatsächlich erzeugten Objekte zu einem bestimmten Zeitpunkt zur Laufzeit dar. Objektdiagramme können als Ergänzung zu den Klassendiagrammen aufgefasst werden.



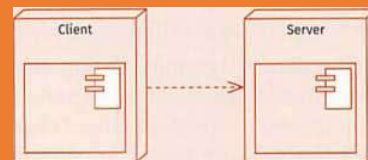
Komponentendiagramm (Component Diagram)

Dieses Diagramm zeigt die Abhängigkeiten und die Organisation der Funktionseinheiten (Komponenten) einer Software. Es wird damit die nächsthöhere Ebene über den Klassen beschrieben.



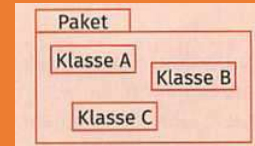
Verteilungsdiagramm (Deployment Diagram)

Dieses Diagramm findet Verwendung bei der Beschreibung von verteilten Systemen, wenn dargestellt werden muss, auf welchen Geräten welche Programme ausgeführt werden und wie diese Programme miteinander kommunizieren. Client-Server-Applikationen, Chat-Systeme oder Online-Spiele basieren beispielsweise auf verteilter Software.



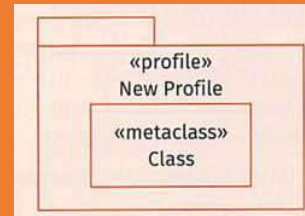
Paketdiagramm (Package Diagram)

Im Paketdiagramm werden Klassen zu Paketen gruppiert. Pakete werden in modernen Programmiersprachen wie Java oder C# eingesetzt, um Klassen übersichtlich anzuordnen.



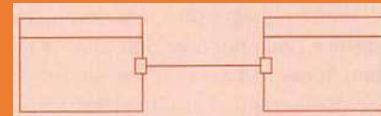
Profildiagramm (Profile Diagram)

Profildiagramme werden auf der Metamodell-Ebene verwendet. Hier werden Stereotype für die Klassen bzw. neue Profile definiert. Das Profildiagramm steht somit über allen anderen UML-Diagrammen, da hier Vorgaben für die weiteren Diagramme erteilt werden können. Praktisch wird aber von dieser Möglichkeit selten Gebrauch gemacht.



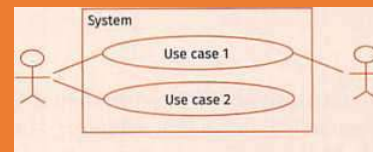
Kompositionsstrukturdiagramm (Composite Structure Diagram)

Das Kompositionsstrukturdiagramm beschreibt die interne Struktur einer Komponente und deren Interaktion mit anderen Komponenten des Systems.



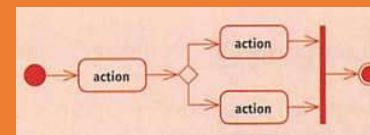
Anwendungsfalldiagramm (Use Case Diagram)

Das Anwendungsfalldiagramm stellt Beziehungen zwischen Akteuren und Aktionen, welche die Akteure auf einem System abrufen können, dar. Es verdeutlicht die grundlegenden Anwendungsszenarien des Systems. Neben dem Klassendiagramm ist das Anwendungsfalldiagramm eines der wichtigsten UML-Diagramme.



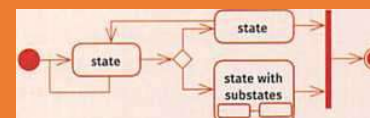
Aktivitätsdiagramm (Activity Diagram)

Das Aktivitätsdiagramm dient zur Beschreibung des Verhaltens einer Klasse oder einer Komponente. Es beschreibt den Ablauf von Aktionen und ist in seinen Darstellungsmöglichkeiten sehr flexibel.



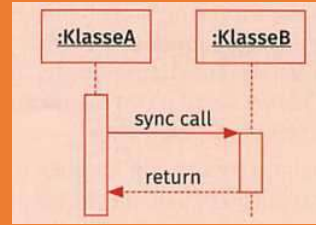
Zustandsdiagramm (State Machine Diagram)

Ein Zustandsdiagramm beschreibt ein System nicht als Aneinanderreihung von Aktionen, sondern als Zustandswechsel. Alle möglichen Zustände, Zustandsübergänge und Ereignisse werden mithilfe dieses Diagramms modelliert. Es wird daher auch als Zustandsautomat bezeichnet.



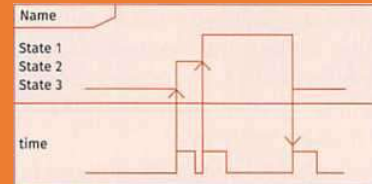
Sequenzdiagramm (Sequence Diagram)

Im Sequenzdiagramm liegt der Fokus auf den Interaktionen zwischen Objekten. Hier wird vor allem der Nachrichtenfluss und Nachrichtenaustausch zwischen den Objekten betrachtet. Dabei spielt vor allem der zeitliche Ablauf der Nachrichten eine entscheidende Rolle. Informationen über die Beziehungen der Objekte untereinander sind im Sequenzdiagramm nicht vorhanden.



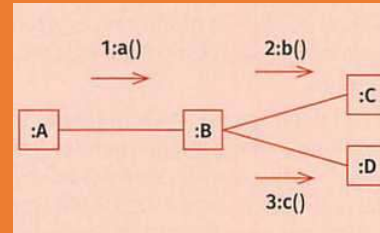
Zeitverlaufsdiagramm (Timing Diagram)

Im Zeitverlaufsdiagramm werden die Zustandswechsel von Objekten innerhalb einer Zeitspanne dargestellt. So kann beispielsweise beschrieben werden, nach wie vielen Sekunden eine Aktion auf eine andere zu erfolgen hat.



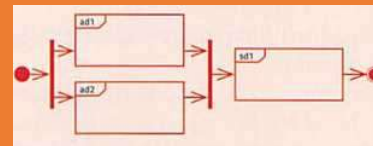
Kommunikationsdiagramm (Communication Diagram)

Mit einem Kommunikationsdiagramm kann dargestellt werden, wie Objekte in einem System oder einer Anwendung zusammenwirken und wie das dynamische Verhalten zwischen diesen ist. Sie sind eine Alternative zum Sequenzdiagramm. Im Sequenzdiagramm liegt das Augenmerk aber auf der zeitlichen Reihenfolge der Nachrichten, im Kommunikationsdiagramm dagegen auf der Struktur der Nachrichten.



Interaktionsübersichtsdiagramm (Interaction Overview Diagram)

Das Interaktionsübersichtsdiagramm vereint mehrere Verhaltensdiagramme, um deren Zusammenspiel darzustellen. So können beispielsweise mehrere Aktivitäts- und Sequenzdiagramme in einem Interaktionsübersichtsdiagramm zusammengefügt werden, um Zusammenhänge zwischen den in den verschiedenen Diagrammen abgebildeten Abläufen darzustellen.



In den nächsten Abschnitten werden die wichtigsten UML-Diagramme und deren Notationselemente anhand von Beispielen näher erläutert. Für die folgende Ausgangssituation werden in diesen entsprechende UML-Diagramme entworfen und beschrieben.

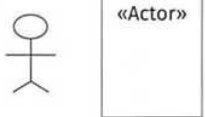
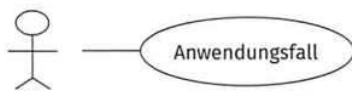
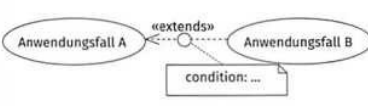
Ausgangssituation für die nachfolgenden UML-Abschnitte (vereinfacht)

Im Rahmen eines Kundenauftrages der Automotik AG soll von der Softwareentwicklungsabteilung des Systemhauses JIKU eine Software für die Wartung von Maschinen und zur Auswertung von Produktionsdaten entwickelt werden. Dabei soll die Software u.a. auch die Wartungsmitarbeiter bei der Behebung von Fehlern unterstützen. Statistiker der Firma sollen die Software zum Anzeigen und zur statischen Auswertung von Produktionsdaten nutzen können. Um eine hohe Softwarequalität des Programms zu gewährleisten, hat sich das Scrum-Team dazu entschieden, bei der Programmentwicklung einen objektorientierten Ansatz zu verfolgen und zur Implementierung dementsprechend auch eine Programmiersprache zu verwenden, welche diesen Ansatz unterstützt. Bei der Entwicklung der Software soll in allen Phasen die moderne Modellierungssprache UML zum Einsatz kommen.

LE 1 - Anwendungsfalldiagramme beschreiben und anwenden

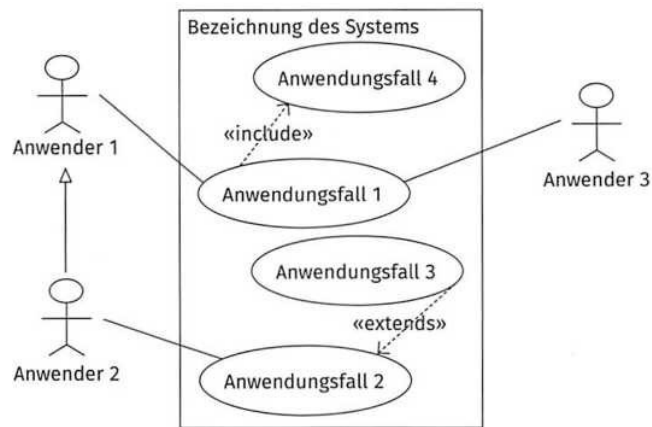
Die Funktionalität eines Programms oder eines Systems lässt sich übersichtlich durch Beschreibung seiner Anwendungsfälle (Use Cases) strukturieren. Diese Anwendungsfälle werden in einem Anwendungsfalldiagramm zusammengefasst. Ein Anwendungsfalldiagramm besteht aus einer Menge von Anwendungsfällen und stellt u.a. die Beziehungen zwischen Akteuren und Anwendungsfällen sowie das äußerlich erkennbare Systemverhalten aus der Sicht eines Anwenders dar. Dabei wird beschrieben, was das System an Anwendungsfällen anbietet und nicht, wie diese zu realisieren sind. Das genaue Verhalten des Systems kann später durch weitere Verhaltensdiagramme genauer beschrieben werden.

Notationselemente und Aufbau

Übersicht über wichtige Notationselemente des Anwendungsfalldiagramms		
Notationselement	Bezeichnung	Beschreibung
	Systemgrenze	Die Systemgrenze umfasst alle Anwendungsfälle, welche das System bereitstellt. Für diese Anwendungsfälle muss später das System die entsprechenden Funktionen bereitstellen.
	Anwender oder Akteur	Hier wird beschrieben, welche externen Benutzer das System verwenden. Dabei ist es auch möglich, dass das System mit anderen externen Systemen interagiert. In diesem Fall wird ein Rechteck statt dem Strichmännchen verwendet.
	Anwendungsfall	Beschreibt eine Aktion oder eine Anzahl von Aktionen, die von dem System für die Benutzer bereitgestellt werden.
	Assoziation	Mit einer Assoziation (Strich) wird eine Beziehung zwischen den Akteuren und den Anwendungsfällen dargestellt.
	Include-Beziehung	Durch eine Include-Beziehung wird die unbedingte Einbindung der Funktionalität eines Anwendungsfalls in einen anderen Anwendungsfall dargestellt. In dem Beispiel schließt der Anwendungsfall A den Anwendungsfall B ein, d.h., wenn A ausgeführt wird, muss auch B ausgeführt werden.
	Extend-Beziehung	Durch eine Extend-Beziehung wird die bedingte Einbindung der Funktionalität eines Anwendungsfalls in einen anderen Anwendungsfall dargestellt. In dem Beispiel kann der Anwendungsfall A durch den Anwendungsfall B erweitert werden, d.h., wenn A ausgeführt wird, kann unter bestimmten Bedingungen auch B ausgeführt werden.

Übersicht über wichtige Notationselemente des Anwendungsfalldiagramms		
Notationselement	Bezeichnung	Beschreibung
	Generalisierung/ Spezialisierung	Eine Generalisierung/Spezialisierung zeigt die Beziehung zwischen einem allgemeinen und einem spezialisierten Element. Eine Generalisierung/ Spezialisierung kann sowohl zwischen Anwendungsfällen als auch zwischen Akteuren dargestellt werden.

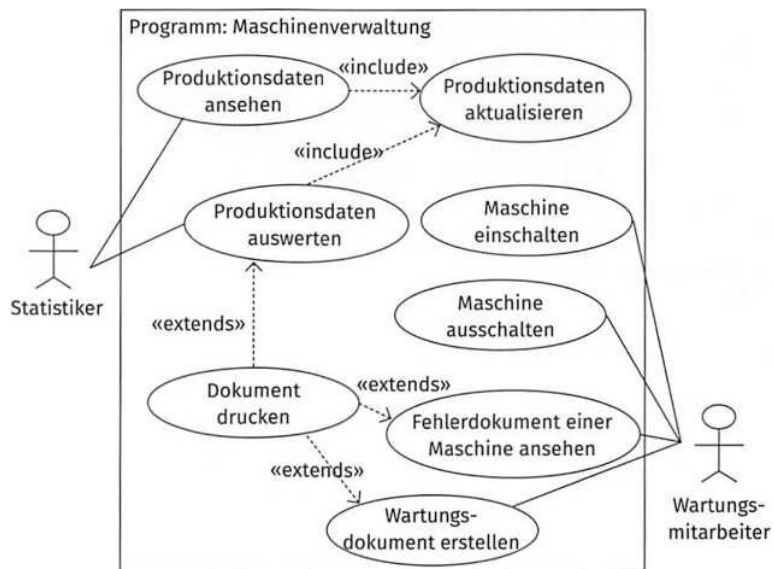
Im Anwendungsfalldiagramm wird das zu beschreibende System als großes Rechteck dargestellt. Zur eindeutigen Bestimmung des Systems wird links oben in dem Rechteck der Systemname vermerkt. Danach wird die Funktionalität des Systems in Form von Anwendungsfällen analysiert und diese innerhalb des Rechtecks dargestellt. Außerhalb des Rechtecks werden die Akteure eingezeichnet, die mit diesem System interagieren. Die Akteure, auch die nicht menschlichen, werden den Anwendungsfällen zugeordnet (assoziiert), die sie ausführen können. Alle Angaben außerhalb des Rechtecks gehören nicht zum Gegenstand der Entwicklungsaufgabe. Im Rahmen der Softwareentwicklung sind jedoch geeignete Schnittstellen vorzuhalten, damit externe Programme auf den betreffenden Anwendungsfall zugreifen können. Allgemein ergeben sich aus den Verbindungen zwischen den Akteuren und den Anwendungsfällen die Schnittstellen, z. B. eine Benutzeroberfläche, welche später im Programm implementiert werden müssen. In der Regel werden für eine zu entwickelnde Software mehrere Anwendungsfalldiagramme erstellt.



LE 2 - Umsetzung eines Anwendungsfalldiagramms für die Ausgangssituation

Für die Ausgangssituation könnte eine Planung für die Anwendungsfälle wie in der Abbildung des Anwendungsfalldiagramms dargestellt aussehen.

Dieses Anwendungsfalldiagramm kann folgendermaßen gelesen werden: Die Wartungsmitarbeiter der Firma sollen mithilfe des Programms Maschinen vom Maschinenpark ein- und ausschalten können. Des Weiteren sollen sie in der Lage sein, sich das Fehlerprotokoll einer Maschine anzusehen oder ein Wartungsprotokoll für diese zu erstellen. Beide Doku-



mente sollen bei Bedarf ausgedruckt werden können. Die Statistiker der Firma sollen sich mit Hilfe des Programms die Produktionsdaten der Maschinen ansehen und statistisch auswerten können. Dazu müssen die Produktionsdaten vom Programm jedes Mal aktualisiert werden. Auch für die Statistiken soll die Möglichkeit des Ausdrucks bestehen. Der Prozess der Wartung der Maschinen und der Auswertung der Produktionsdaten wurde hier sehr vereinfacht dargestellt. Für eine komplexere Aufgabenstellung würde auch das Anwendungsfalldiagramm komplexer ausfallen. Alternativ können auch mehrere Anwendungsfalldiagramme dann erstellt werden.

Weitere Beispielaufgabe zum Anwendungsfalldiagramm

Beispielaufgabe

Im Rahmen eines Kundenauftrages soll ein Programm für eine Onlinereservierung von Hotelzimmern für eine Hotelkette entwickelt werden. In einem ersten Gespräch sind schon einige Informationen über die Funktionalität des Programms erörtert worden. So sollen die Kunden u.a. Zimmer reservieren können. Der Reservierungsvorgang schließt immer mit ein, dass das Programm nochmal prüft, ob das entsprechende Zimmer zur Verfügung steht. Die Reservierung schließt weiterhin die Angabe einer Zahlungsmethode mit ein. Wird die Zahlungsmethode „Kreditkartenzahlung“ gewählt, dann soll das Programm die Kreditkartendaten bei der entsprechenden Bank überprüfen. Dabei stellt die Bank ein anderes externes System dar, welche mit der Anwendung interagiert. Sie sollen aus diesen Angaben ein Anwendungsfalldiagramm entwickeln, welches als Diskussionsgrundlage für weitere Treffen mit dem Auftraggeber angedacht ist.

Lösung

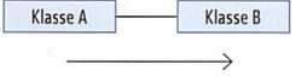
Zur Lösung wird zunächst einmal das System als Rechteck gezeichnet und ein entsprechender Name vergeben. Danach werden die Akteure und die Anwendungsfälle laut Aufgabenstellung analysiert und in das Diagramm eingezeichnet. In diesem Fall sind die Akteure die Kunden, welche das Programm benutzen und die Bank als ein anderes externes System, welches mit dem Programm interagiert. Zu den Anwendungsfällen zählen „Zimmer reservieren“, „Verfügbarkeit prüfen“, „Zahlungsmethode eingeben“ und „Kreditkarte prüfen“. Danach werden die Akteure und die Anwendungsfälle in Beziehung gesetzt. Das entsprechende Anwendungsfalldiagramm sieht dann wie folgt aus.

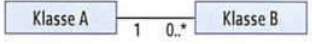


TAG 7 - Klassendiagramme beschreiben und anwenden

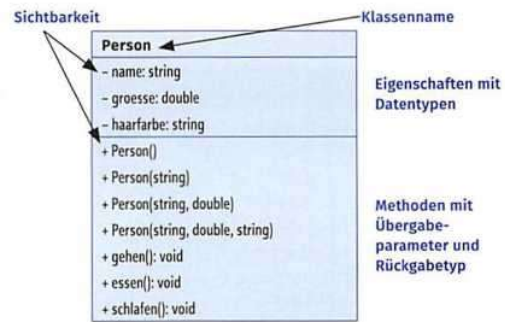
Klassendiagramme bilden ein zentrales Element der UML. Eingesetzt werden sie vor allem bei der Analyse und beim Entwurf eines Softwaresystems. Klassendiagramme können unterschiedliche Detaillierungsgrade aufweisen und beschreiben grafisch die Beziehungen zwischen den Klassen einer Anwendung. Klassendiagramme geben eine statische Sichtweise auf die Anwendung wieder und zählen somit zu den Strukturdiagrammen. Heutzutage haben viele moderne IDE schon Editoren integriert, mit deren Hilfe sich Klassendiagramme erstellen lassen. Oder sie können über Extensions um die entsprechende Funktionalität erweitert werden.

LE 1 - Notationselemente und Aufbau

Übersicht über wichtige Notationselemente des Klassendiagramms				
Notationselement	Bezeichnung	Beschreibung		
	Klasse	Klassen werden in der UML durch Rechtecke dargestellt, die den Namen der Klasse und/oder die Eigenschaften und Methoden der Klasse enthalten. Klassenname, Eigenschaften und Methoden werden durch eine horizontale Linie getrennt. Der Klassenname steht im Singular und beginnt mit einem Großbuchstaben. Eigenschaften können näher beschrieben werden, z. B. durch ihren Typ. Methoden können ebenfalls durch Parameter usw. ergänzt werden.		
	Alternative Klassendarstellung	Bei den alternativen Klassendarstellungen werden meistens die Eigenschaften und Methoden weggelassen. Sie dienen vor allem dazu, die Beziehungen zwischen den einzelnen Klassen zu modellieren.		
+, -, #, ~ + schwimmen(): void	Sichtbarkeit für Eigenschaften und Methoden	Mit der Sichtbarkeit werden die Zugriffsmodifier der OOP beschrieben	+	public
			-	private
			#	protected
	Assoziation gerichtete Assoziation	Eine Assoziation ist eine Beziehung zwischen zwei Klassen. Es ist keine Teil-Ganzes-Beziehung. Gerichtete Assoziationen werden durch einen zusätzlichen Pfeil gekennzeichnet.		
	Aggregation	Eine Aggregation ist eine Beziehung zwischen zwei Klassen, wobei es sich um eine Teil-Ganzes-Beziehung handelt. Die Aggregation wird durch eine Verbindungslinie zwischen den Klassen mit einer nicht ausgefüllten Raute am Ende zur „Ganzes“-Klasse beschrieben.		
	Komposition	Eine Komposition ist eine Beziehung zwischen zwei Klassen, wobei es sich um eine „starke“ Teil-Ganzes-Beziehung handelt. Die Komposition wird durch eine Verbindungslinie zwischen den Klassen mit einer ausgefüllten Raute am Ende zur „Ganzes“-Klasse beschrieben.		

Übersicht über wichtige Notationselemente des Klassendiagramms		
Notationselement	Bezeichnung	Beschreibung
	Multiplizität	Die Multiplizität gibt an, wie viele Objekte der Klasse A mit wie vielen Objekten der Klasse B in Beziehung stehen können. Die Angaben können Assoziationen, Aggregationen oder Kompositionen zugeordnet werden.
	Vererbung	In der UML wird die Vererbung durch eine Linie mit einer dreieckigen, nicht ausgefüllten Pfeilspitze dargestellt, die von der Unterklasse zur Oberklasse weist. Wird von einem Interface geerbt, dann wird anstatt einer durchgehenden Linie eine Strichlinie verwendet.

Einzelne Klassen werden i.d.R. mit allen Eigenschaften und Methoden und deren Sichtbarkeiten dargestellt. Bei den Eigenschaften wird zusätzlich noch der Datentyp und bei den Methoden die Übergabeparameter und der Rückgabtyp mit angegeben. Weiterhin können noch weitere Informationen der Klasse in Form von Schlüsselwörtern (interface, abstract etc.) hinzugefügt werden. Des Weiteren werden in einem Klassendiagramm die Beziehungen, welche zwischen den Objekten der einzelnen Klassen bestehen, modelliert. Dazu gibt es verschiedene Möglichkeiten, eine Beziehung zu beschreiben. Die Notationen dafür wurden schon in der vorhergehenden Tabelle aufgeführt. Diese werden anhand eines Beispiels nochmals genauer erläutert.



Beziehungen zwischen Klassen

Assoziation

Die Assoziation gibt an, dass eine Verbindung zwischen den Objekten von zwei Klassen besteht.

Beispiel: Eine Maschine erstellt oder bearbeitet ein Werkstück. Es besteht also eine Beziehung. Allerdings ist die Beziehung keine Teil-Ganzes-Beziehung.



Aggregation

Die Aggregation ist eine Variante der Assoziation. In diesem Fall ist es eine Teil-Ganzes-Beziehung. Bei der Aggregation können die einzelnen Objekte aber auch unabhängig voneinander existieren. Die "Ganzes"-Klasse wird auch als Aggregationsklasse bezeichnet.

Beispiel: Die Aggregationsklasse ist das Auto. Die Räder sind ein Bestandteil eines Autos, können aber auch abmontiert werden und unabhängig vom Auto existieren.



Komposition

Die Komposition ist eine Variante der Aggregation. Die Komposition erfordert einen starken Besitz über die Teilobjekte. Der starke Besitz hat Auswirkungen auf den Umgang mit dem Aggregationsobjekt. Wird z.B. das Ganze gelöscht, werden die Teile auch gelöscht. Das heißt, die Teile können nicht ohne das Ganze existieren.

Beispiel: Das Ganze ist in diesem Fall das Haus und die Räume sind ein Bestandteil des Hauses. Wird das Haus abgerissen, existieren auch die Räume des Hauses nicht mehr.



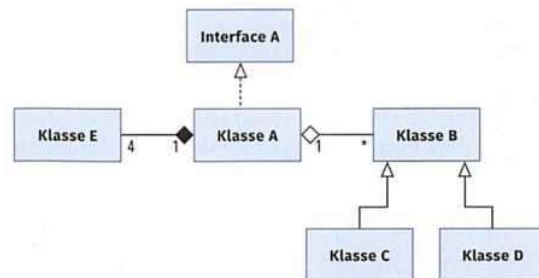
Vererbung

Bei der Vererbung werden aus einer bestehenden Klasse eine oder mehrere neue Klassen abgeleitet. Die Eigenschaften einer Oberklasse werden an die zugehörige Unterklasse weitergegeben (siehe auch S. 186).

Beispiel: Aus der Basisklasse „Wasserfahrzeug“ werden die Klassen „Segelboot“ und „Motorboot“ abgeleitet.



Sollen die Beziehungen zwischen den Klassen einer Software schwerpunktmäßig dargestellt werden, dann wird meistens auf die Angaben der Eigenschaften und Methoden der einzelnen Klassen verzichtet. Dadurch wird eine übersichtliche Darstellung erreicht, welche sich auf das Wesentliche konzentriert (siehe auch das nebenstehende abstrakte Beispiel). Bei den Assoziationen, Aggregationen und Kompositionen werden dann noch die Multiplizitäten angegeben.

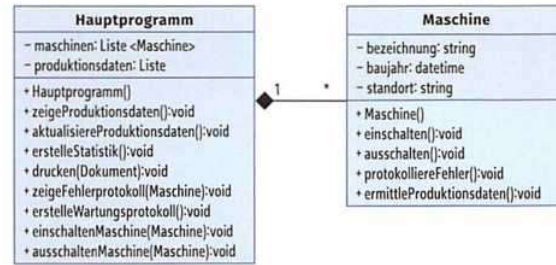


Beispiele für Multiplizität

1	Es gibt genau ein Objekt.
1,3,8	Es gibt entweder ein, drei oder acht Objekte.
2 .. 7	Die Anzahl der Objekte kann von zwei bis sieben variieren.
0 .. * oder *	Es kann eine beliebige Anzahl an Objekten geben.

LE 2 - Umsetzung eines Klassendiagramms für die Ausgangssituation

In einem Klassendiagramm für die Ausgangssituation könnten beispielsweise die Klassen „Maschine“ und „Hauptprogramm“ mit ihren Eigenschaften, Methoden und den entsprechenden Sichtbarkeiten genau beschrieben und die Beziehungen zwischen diesen dargestellt werden.



Die Methoden der Klassen wurden direkt aus dem Anwendungsfalldiagramm abgeleitet. Auf die Angabe der Get- und Set-Methoden wurde bei allen Klassen bewusst verzichtet und nur die Arbeitsmethoden dargestellt. Die beiden Klassen sind in Form einer Komposition miteinander verbunden. Diese Beziehung wurde gewählt, weil beliebig viele Maschinen Bestandteil des Hauptprogramms in Form einer dynamischen Liste sein können. Wenn das Hauptprogramm gelöscht wird, dann werden auch alle Objekte der Maschinen gelöscht. Auch hier ist wieder nur eine vereinfachte Variante dargestellt. Es ist mit Sicherheit davon auszugehen, dass bei den einzelnen Klassen noch weitere Eigenschaften und Methoden für komplexere Aufgabenstellungen geplant werden müssen. Eine andere mögliche Erweiterung könnte beispielsweise die Bildung von Unterklassen (Spezialisierungen) der Klasse „Maschine“ sein. In diesem Fall müssten eventuell auch einige Sichtbarkeiten von *private* (-) auf *protected* (#) geändert werden.

LE 3 - Weitere Beispielaufgabe zum Klassendiagramm

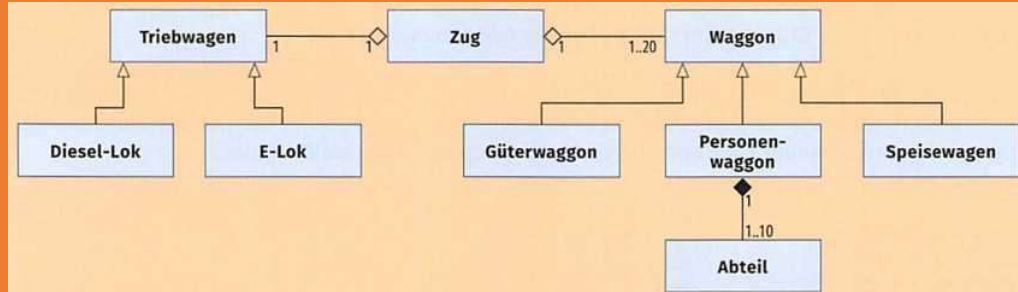
Beispielaufgabe

Im Rahmen eines Kundenauftrages für ein privates Bahnunternehmen soll ein Programm erstellt werden, das u.a. Züge verwaltet. Ein Zug (hier eine abstrakte Planungseinheit im Programm) wird für jeweils eine Fahrt aus verschiedenen externen Objekten zusammengestellt. Einem Zug werden jeweils ein Triebwagen und von einem bis zu zwanzig Waggons zugeordnet. Als Triebwagen kommen Diesel-Loks und E-Loks in Betracht. Bei den Waggons gibt es folgende Arten: Güterwaggons, Personenwaggons und Speisewagen. Dabei kann ein Personenwaggon bis zu zehn Abteile besitzen. Erstellen Sie ein Klassendiagramm, in dem die Beziehungen zwischen den Klassen abgebildet werden. Auf die Angabe von Eigenschaften und Methoden für die einzelnen Klassen kann verzichtet werden.

Lösung

Für die Lösung müssen die in der Aufgabenstellung enthaltenen Objekte ermittelt und dafür Klassen modelliert und anschließend sinnvoll miteinander in Beziehung gesetzt werden. Objekte wären: Zug, Triebwagen, Waggon, Diesel-Lok, E-Lok, Güterwaggon, Personenwaggon, Speisewagen und das Abteil. Daraus können gleichlautende Klassen gebildet und in Beziehung gesetzt werden. Da ein Zug aus Waggons und Triebwagen besteht, handelt es sich um eine Teil-Ganzes-Beziehung. In diesem Fall ist es eine Aggregation, weil die Triebwagen und Waggons auch ohne den Zug weiter existieren und nur diesem für einen gewissen Zeitraum zugeordnet werden.

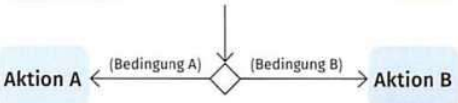
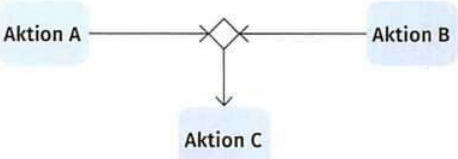
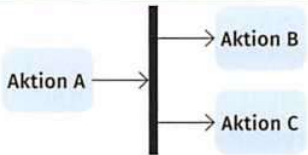
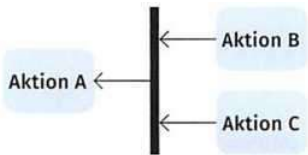
Diesel-Lok und E-Lok sind spezielle Triebwagen und stehen mit dieser Klasse in Form einer Vererbung in Verbindung. Dasselbe gilt für Waggon, Güterwaggon, Personenwaggon und Speisewagen. Die Beziehung zwischen Personenwaggon und Abteil ist eine Komposition, weil die Abteile nur existieren, solange der Personenwagen existiert.

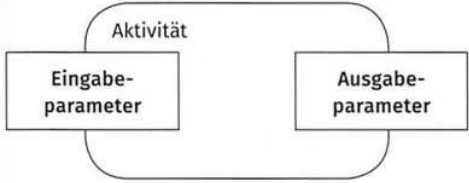


TAG 8 - Aktivitätsdiagramm beschreiben und anwenden

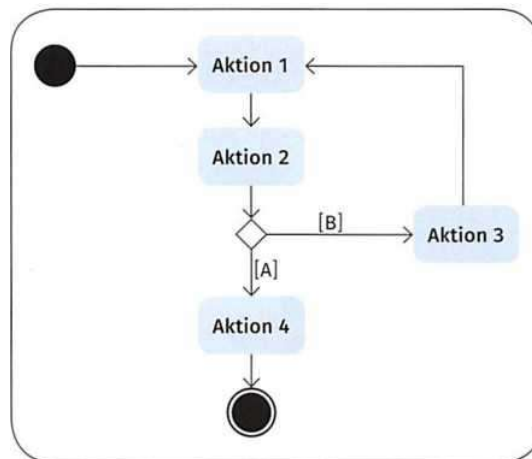
Während Diagramme wie das Klassendiagramm eine statische Sicht auf das System abbilden, kann mithilfe vom Aktivitätsdiagramm eine dynamische Sicht bzw. das Verhalten des Systems modelliert werden. Das Aktivitätsdiagramm beschreibt den Ablauf von Aktionen und verwendet wenige Symbole für sehr umfangreiche Darstellungsmöglichkeiten. Es kann hinsichtlich seiner Darstellungsweise und seiner Aussagen mit einem Programmablaufplan verglichen werden. Aktivitätsdiagramme werden in vielen Phasen der Softwareentwicklung verwendet. In der Analysephase wird es beispielsweise eingesetzt, um Geschäftsprozesse und Tätigkeiten zu beschreiben. In der Entwurfsphase wird es in vielen Fällen zur Darstellung des zu entwickelnden Algorithmus verwendet. In der Testphase dienen sie als Grundlage, um Testszenarien zu entwickeln und sie eignen sich hervorragend, um aussagekräftige Entwicklungsdokumentationen zu erstellen.

LE 1 - Notationselemente und Aufbau

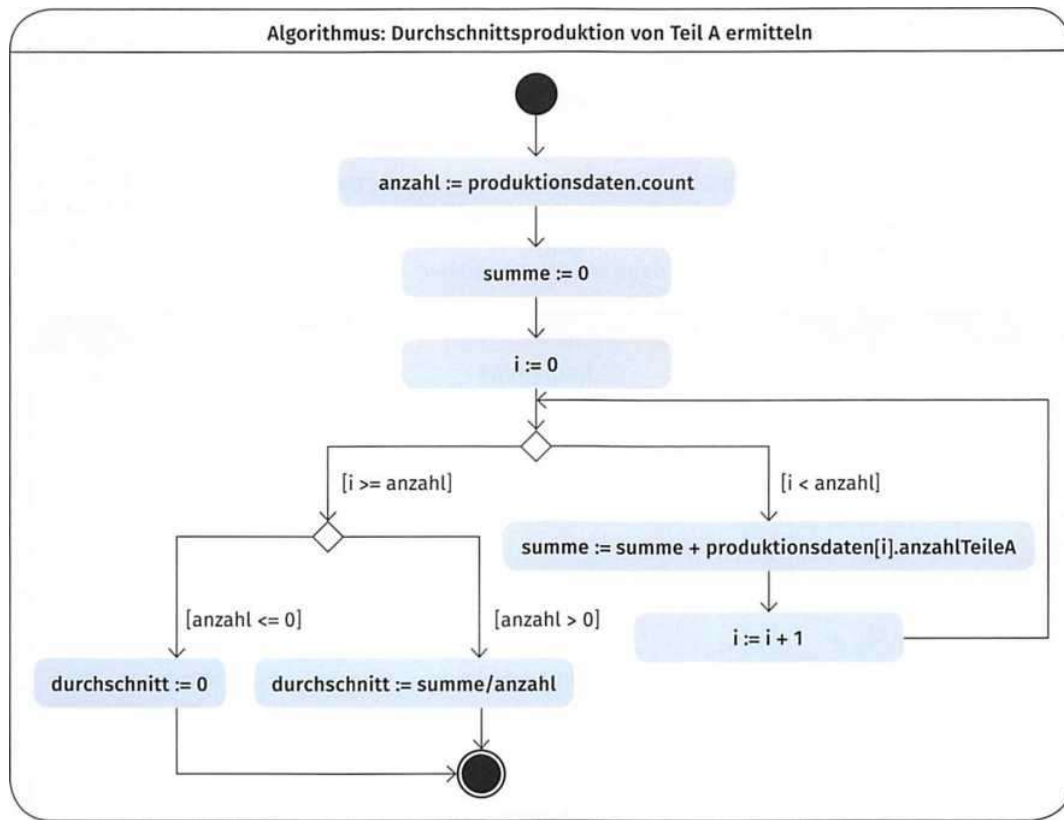
Übersicht über wichtige Notationselemente des Aktivitätsdiagramms		
Notationselement	Bezeichnung	Beschreibung
	Startknoten Endknoten Ablaufende	Der Startknoten markiert den Beginn eines Ablaufes. Der Endknoten markiert das Ende eines Ablaufes. Das Ablaufende markiert das Ende eines Zweiges.
	Aktion	Sie beschreibt das Verhalten, welches eine Veränderung herbeiführt.
	Kante	Sie dient Zur Angabe des Kontrollflusses zwischen den einzelnen Aktionen.
	Verzweigung	Je nachdem, ob die Bedingung A oder die Bedingung B erfüllt ist, wird die Aktion A oder die Aktion B ausgeführt.
	Zusammenführung	Nach dem Ende der Aktion A oder der Aktion B wird die Aktion C ausgeführt.
	Aufspaltung	Nach dem Ende einer Aktion (A) werden mehrere Aktionen (B und C) gleichzeitig begonnen und parallel ausgeführt.
	Synchronisation	Nach dem Ende mehrerer Aktionen (B und C) wird eine Aktion (A) ausgeführt. Erst wenn B und C beendet sind, wird A ausgeführt.

Übersicht über wichtige Notationselemente des Aktivitätsdiagramms		
Notationselement	Bezeichnung	Beschreibung
	Aktivität	Innerhalb dieser Grenze werden die einzelnen Aktionen und deren Kontrollfluss dargestellt. Aktivitäten können zusätzlich noch in verschiedene Partitionen unterteilt werden. Eine Aktivität kann Ein- und Ausgabeparameter haben, welche als Rechteck dargestellt werden.

Im Aktivitätsdiagramm wird die zu beschreibende Aktivität als Rechteck dargestellt. Zur eindeutigen Bestimmung der Aktivität wird links oben im Rechteck die Aktivitätsbezeichnung vermerkt. Danach wird der zu beschreibende Prozess oder Algorithmus innerhalb dieses Rechteckes modelliert. Dabei kann eine Aktivität in verschiedene Partitionen unterteilt werden und Ein- und Ausgabeparameter besitzen. Die Unterteilung in verschiedene Partitionen kann beispielsweise erfolgen, wenn ein Prozess beschrieben wird, an dem mehrere Personen beteiligt sind. Für jede Person wird dann eine Partition angelegt und die Aktionen, welche innerhalb des Prozesses von diesen Personen ausgeführt werden, der entsprechenden Partition zugeordnet.



LE 2 - Umsetzung eines Aktivitätsdiagramms für die Ausgangssituation



In dem abgebildeten Aktivitätsdiagramm für die Ausgangssituation wird der Algorithmus zur Berechnung der Durchschnittsproduktion von einem Teil A modelliert. Dabei werden zunächst die Variablen initialisiert. Danach werden aus der Liste mit den Produktionsdaten die produzierten Teile von A innerhalb einer Schleife aufsummiert. Wenn die Schleife verlassen wird, dann wird der Durchschnitt aus der Summe und der Anzahl berechnet. Davor muss allerdings sichergestellt werden, dass keine Division durch null auftritt. Dieses wäre immer dann der Fall, wenn keine Produktionsdaten in der Liste vorhanden sind. Die Variable „anzahl“ hat in diesem Fall den Wert Null und die Schleife wird nicht durchlaufen.

Weitere Beispielaufgabe zum Aktivitätsdiagramm

Beispielaufgabe

Im Rahmen eines Kundenauftrages für ein Gastronomieunternehmen soll ein Programm entwickelt werden. In der Analysephase dieses Projektes soll u.a. der Prozess eines Restaurantbesuches für spätere Betrachtungen mithilfe eines Aktivitätsdiagramm erfasst und dokumentiert werden. Das Gastronomieunternehmen beschreibt einen typischen Restaurantbesuch wie folgt:

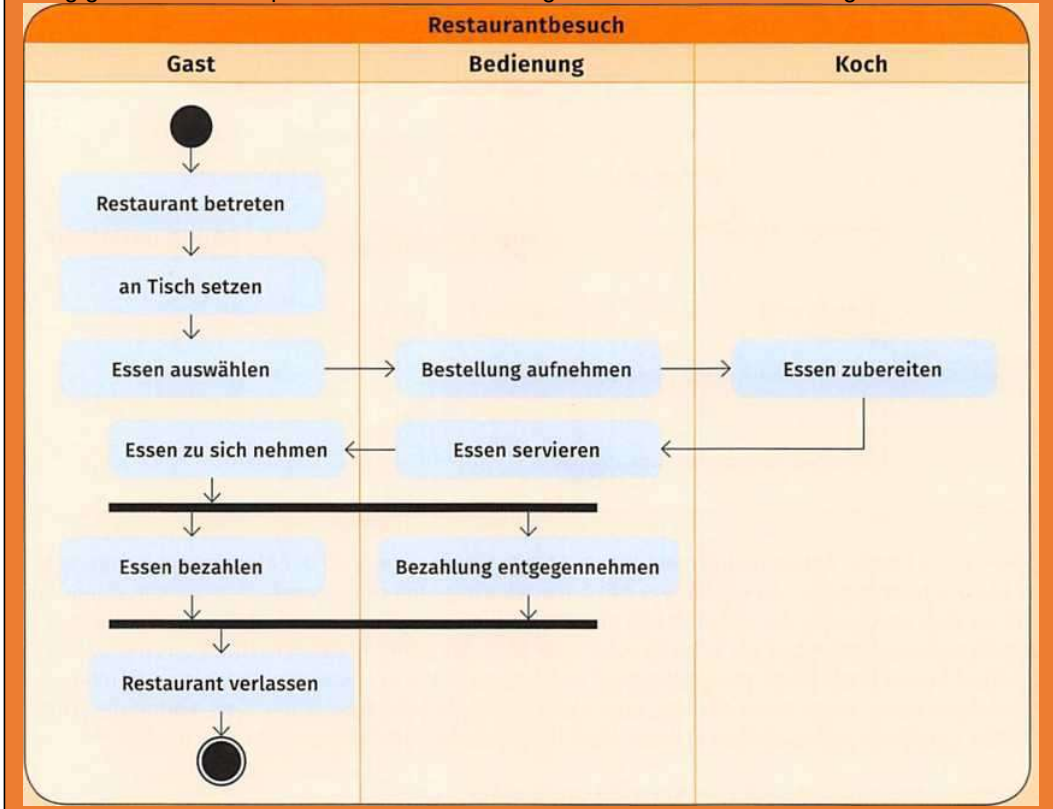
An dem Prozess als solches sind neben dem Gast auch noch die Bedienung und der Koch beteiligt und folgende Aktionen werden während des Prozesses ausgeführt:

- Gast: Restaurant betreten, an Tisch setzen, Essen auswählen, Essen zu sich nehmen, Essen bezahlen, Restaurant verlassen
- Bedienung: Bestellung aufnehmen, Essen servieren, Bezahlung entgegennehmen
- Koch: Essen zubereiten

Zeichnen Sie ein Aktivitätsdiagramm und modellieren Sie den Prozess des Restaurantbesuchs, indem Sie die genannten Aktionen sinnvoll miteinander in Beziehung setzen. Dabei soll beachtet werden, dass das Bezahlen und die Bezahlung entgegennehmen ein interaktiver Prozess ist, welcher parallel abläuft. Die Möglichkeit, dass der Gast jederzeit das Restaurant wieder verlassen kann, soll in diesem Diagramm nicht dargestellt werden.

Lösung

Zur Lösung wird zunächst einmal für die Aktivität ein Rahmen gezeichnet und eine entsprechende Bezeichnung vergeben. Danach wird die Aktivität in drei Partitionen (Gast, Bedienung, Koch) aufgeteilt, die Aktionen zugeordnet und danach sinnvoll miteinander in Beziehung gesetzt. Ein entsprechendes Aktivitätsdiagramm könnte dann wie folgt aussehen:



TAG 9 - Sicherheitsrelevante Aspekte bei der Softwareplanung berücksichtigen

Die Gefährdung von Daten und Software nimmt im Rahmen der Digitalisierung immer mehr zu, besonders, wenn diese über das Internet zugänglich sind. Besonders kritisch wird es, wenn sensible Daten oder geschäftsrelevante Software gezielt angegriffen werden. Um die digitale Infrastruktur in den Unternehmen, aber auch im privaten Bereich zu schützen, stehen verschiedenste Möglichkeiten zur Verfügung.



In diesem Abschnitt werden Möglichkeiten betrachtet, die Informationssicherheit schon bei der Entwicklung der Software zu gewährleisten. Hierbei stehen die Entwicklerinnen und Entwickler der Software in der Verantwortung. Von ihnen müssen Konzepte entwickelt und umgesetzt werden, welche die Informationssicherheit vor, während und nach der Verarbeitung von Daten sicherstellen.

Security by Design

Secure by Design (SBD) bedeutet in der Softwareentwicklung, dass Softwareprodukte von Grund auf so konzipiert wurden, dass sie sicher sind.

Privacy by Design

Privacy by Design erfordert, dass der Datenschutz während des gesamten Engineering-Prozesses berücksichtigt wird.

Die Grundlage dafür gibt die DSGVO der EU vor. Bei Nichteinhaltung dieser Regelungen ist teilweise mit erheblichen finanziellen Einbußen zu rechnen. Außerdem entsteht meistens bei sicherheitsrelevanten Mängeln der Software ein Imageschaden für die Marke und ein Vertrauensverlust der Kunden. Nach den Methoden „Security by Design“ und „Privacy by Design“ der DSGVO sollen spezielle Schutz- und Vorsorgemaßnahmen eingehalten werden, bei denen Vertraulichkeit, Verfügbarkeit und Integrität der Daten im Mittelpunkt stehen. So müssen beispielsweise sensible Daten vor unberechtigten Zugriffen, Manipulation und Löschung geschützt werden. Um einen soliden Schutz der Daten und der Software zu gewährleisten, ist es erforderlich, dass die Informationssicherheit nicht erst zum Ende hin, sondern während des gesamten Entwicklungsprozesses der Software berücksichtigt wird.

Um eine sichere Software zu entwickeln, hat sich ein mehrschichtiges Modell für einen sicheren Umgang mit Daten bewährt. Dabei gelten für jede Schicht einer Softwarearchitektur und der dazugehörigen technischen Infrastruktur eigene Sicherheitsmaßnahmen. Die einzelnen Schichten bauen von innen nach außen aufeinander auf, sodass die äußeren Schichten eventuelle Sicherheitslücken der inneren Schichten kompensieren und somit das Gefahrenrisiko minimieren. Sicherheitsmaßnahmen, welche die technische Infrastruktur betreffen, werden u.a. im Lernfeld 9 behandelt. Sicherheitsmaßnahmen im Bereich der Softwarearchitektur müssen unbedingt die Bereiche der Zugriffskontrolle, Authentifizierung, der Datenvalidierung (Ein- und Ausgabe), der Fehlerbehandlung, Protokollierung und Kryptographie umfassen. Dazu sind in der Designphase sechs Sicherheitsprinzipien zu beachten.

Sechs Sicherheitsprinzipien	
Minimalprinzip	Die Benutzer der Software sollten nur die Berechtigungen bekommen, welche sie für die Erledigung ihrer Aufgaben wirklich benötigen (z. B. nur die Berechtigungen zum Abfragen von Daten, aber nicht zum Bearbeiten).
Separierung von Berechtigungen	Bestimmte Benutzer bekommen nur den Zugriff auf bestimmte Funktionen in der Software (z. B. Hinzufügen oder Löschen von Daten).
Vollständige Zugriffsüberwachung	Jeder Zugriff auf die Software sollte überprüft werden, um unberechtigten Zugriff zu verhindern.
Mehrere Sicherheitsebenen	Hierbei wird die Software in mehrere Sicherheitsebenen aufgeteilt. Dadurch wird die Gefahr von Single Point of Failure (SPoF) hinsichtlich der Sicherheit verringert. Je mehr Sicherheitsebenen die Software hat, desto weniger können eventuelle Schwachstellen ausgenutzt werden.
Sicherer Ausfall	Wenn eine Software ausfällt, sollte sie trotzdem einen sicheren Zustand in den Bereichen der Vertraulichkeit und Integrität aufweisen. Deswegen sollte auch die fertige Software mit sicheren Standardeinstellungen ausgeliefert werden.
Benutzerfreundliche Sicherheit	Die Sicherheitsmechanismen sollten immer benutzerfreundlich gestaltet sein, um das Benutzererlebnis nicht zu beeinträchtigen. Im anderen Fall ist die Gefahr groß, dass die Benutzer die Sicherheitsmaßnahmen einfach deaktivieren (siehe auch UX Design).

Für die Umsetzung der Informationssicherheit bei Software Produkten stehen aktuell viele geeignete Frameworks zur Verfügung, welche eine eigene Implementierung überflüssig machen. Deshalb sollten bei der Auswahl der Programmiersprache und der Frameworks auch die zur Verfügung stehenden Technologien hinsichtlich der Informationssicherheit berücksichtigt werden.

Design- und Umsetzungshinweise für sichere Software

1. Trennung von Datenverarbeitung und -darstellung in einzelne Schichten

Designüberlegungen: Hierbei wird das Programm in verschiedene separate Einheiten aufgeteilt, welche unterschiedliche Funktionen wie die Verarbeitung oder die Darstellung der Daten übernehmen. Für die Umsetzung wird häufig das MVC-Pattern (Model-View-Controller) verwendet, durch welches die Anwendung in drei separate Bereiche aufgeteilt wird.

Umsetzungsmöglichkeiten: Einsatz von Frameworks, welche das MVC-Pattern unterstützen und Werkzeuge für die Erstellung von Controllern, Views und Models bereitstellen

2. Validierung der Daten vor der Verarbeitung

Designüberlegungen: Eine konsequente Überprüfung von Eingabe- und Ausgabedaten stellt ein Grundprinzip für die Entwicklung von stabilen und sicheren Anwendungen dar. Ohne diese Überprüfung besteht die Gefahr, dass unvollständige oder sogar inkorrekte Daten, z. B. ein negatives Gewicht, verarbeitet und gespeichert werden, was u.a. auch eine geringe Qualität des Datenbestandes nach sich ziehen kann.

Umsetzungsmöglichkeiten:

Überprüfung der Daten an verschiedenen Stellen der Anwendung, bevor sie weiterverarbeitet werden, z. B. Ein- und Ausgabedaten, Übergabeparameter von Methoden usw.
Richtige Datentypen und Wertebereiche wählen

3. Sichere Übertragung von Daten gewährleisten

Designüberlegungen: Eine verschlüsselte Übertragung von Daten stellt sicher, dass ein Angreifer keinen Zugang zu den Daten bekommt, auch wenn er sich bereits in den Datenaustausch eingeklinkt hat, z. B. Man-in-the-Middle-Angriff. Über das HTTP(S)-Protokoll verschlüsselte Internetverbindungen liefern dabei einen hohen Sicherheitsstandard.

Umsetzungsmöglichkeiten: Verschlüsselung nicht selbst implementieren, sondern auf die Möglichkeiten der Frameworks für Übertragungsverschlüsselung zurückgreifen

4. Sichere Benutzerverwaltung gewährleisten

Designüberlegungen: Umsetzung einer vollständigen und robusten Berechtigungsprüfung

Umsetzungsmöglichkeiten: Für sichere Implementierung der Login-Funktionalität Standardverfahren der entsprechenden Frameworks verwenden

5. SOAP und REST-Schnittstellen sichern

Designüberlegungen: Häufig stellen Anwendungen Teile ihrer Funktionalität über REST- oder SOAP-Schnittstellen bereit. Je nachdem, welche Datenformate diese Schnittstellen verwenden, benötigen sie Filterroutinen, welche spezifische Sicherheitsfunktionen für das jeweilige Format ergänzen.

Umsetzungsmöglichkeiten:

- Überprüfen und Filtern aller Daten anhand einer Whitelist, welche eine Liste der zulässigen Werte und Zeichenketten enthält
- Nutzen der Möglichkeiten von Frameworks zum Filtern oder Whitelisting

6. Entwicklung und Einsatz von Unittests

Designüberlegungen: Schon während der Entwicklung erfolgt ein regelmäßiger Einsatz von Tests mit entsprechenden Testdaten.

Umsetzungsmöglichkeiten:

- Erstellung eines Unittests für jede Funktion
- Nutzung automatisierter Unittests

7. SQL-Injections verhindern

Designüberlegungen: SQL-Injections entstehen, wenn Benutzereingaben zur Erstellung von SQL-Anweisungen ohne ausreichende Überprüfung verwendet werden. Durch Manipulation der Eingabesyntax führt ein Angreifer unerwünschte Befehle aus und erhält dadurch beispielsweise unbefugten Zugriff auf Daten.

Umsetzungsmöglichkeiten: Nutzung der Möglichkeiten des eingesetzten Frameworks zur Verhinderung von SQL-Injections

8. Protokollierung von sicherheitsrelevanten Ereignissen

Designüberlegungen: Tritt ein Sicherheitszwischenfall ein oder treten Fehler auf, sind Support- und Entwicklungsteams auf möglichst detaillierte Informationen angewiesen. Anderenfalls lassen sich die Vorfälle nicht nachvollziehen und die Ursache beseitigen. Dazu sollten diese Ereignisse von der Software protokolliert werden. Folgende Ereignisse können u.a. protokolliert werden: fehlerhafte Anmeldeversuche, Fehler in der Anwendung (Exceptions), Fehler bei der Datenvalidierung, Änderungen bei Berechtigungen, Administratoraktivitäten, Schreibzugriffe auf Daten usw.

Umsetzungsmöglichkeiten:

- Erstellen von Log-Dateien oder Protokollen
- Einhaltung der aktuellen Datenschutzbestimmungen bezüglich der Protokollierung personenbezogener Daten

Nachfolgend wird beispielhaft aufgezeigt, wie die Informationssicherheit im Rahmen der Softwareentwicklung mit Scrum geplant und umgesetzt werden kann.

Exemplarische Umsetzung der Informationssicherheit im Scrum-Prozess

1. Ermitteln der sicherheitsrelevanten Aspekte

Dokumente: User Story

Inhalte:

- Analyse der Art und der Qualität der Daten
- Analyse der Bedrohungslage der Software
- Risikobewertung usw.

Verantwortlichkeit: Product Owner, Datenschutzbeauftragte

2. Planung sicherheitsrelevanter Maßnahmen

Dokumente: Product Backlog

Inhalte:

- datenschutzfreundliche Voreinstellungen
- Datenverschlüsselung muss vorgesehen werden
- Festlegung von Speicherfristen
- Protokollierung von Zustimmungen
- Rechte-/Rollenkonzept usw.

Verantwortlichkeit: Product Owner, Datenschutzbeauftragte

3. Implementieren der sicherheitsrelevanten Aspekte in der Software

Dokumente: Sprint Backlog

Maßnahmen bei der Umsetzung:

anerkannte Richtlinien zur sicheren Softwareentwicklung werden berücksichtigt
Tests werden regelmäßig nach den geplanten Testszenarien durchgeführt
wann immer möglich werden Testdaten verwendet und keine Produktivdaten
Einsatz anerkannter Verschlüsselungsverfahren anstatt einer Eigenentwicklung usw.

Verantwortlichkeit: Entwicklungsteam

4. Abnahme der Software unter sicherheitsrelevanten Gesichtspunkten

Dokumente: Definition of Done, Abnahmeprotokoll

Inhalte: Teil des Akzeptanzprozesses ist auch die Prüfung, ob die Anforderungen im Bereich der Informationssicherheit korrekt umgesetzt wurden. Dabei ist zu überprüfen, ob das Product Backlog ggf. ergänzt oder modifiziert werden muss.

Verantwortlichkeit: Product Owner

LE 1 - Benutzerschnittstellen unter softwareergonomischen Gesichtspunkten planen

Von besonderer Bedeutung für die Softwarequalität ist die Softwareergonomie.

Softwareergonomie

Die Softwareergonomie ist ein Bereich der Informatik, welcher sich mit der Anpassung von Software an die Bedürfnisse und Erfordernisse des Benutzers befasst. Gegenstand der Anpassung ist dabei die sogenannte Benutzerschnittstelle (User Interface oder Benutzungsoberfläche) über die der Mensch mit dem Computer interagiert.



Das Spektrum der Interaktionen reicht dabei von Benutzeroberflächen mit einfachen alphanumerischen Eingabemöglichkeiten über mit der Maus zu bedienende GUIs bis hin zu Interaktionsformen mit virtueller Realität. Durch immer neue Technologien ergeben sich auch ständig neue Interaktionsformen und damit verändern sich auch die Anforderungen an die Softwareergonomie. Bei Interaktionsformen wie der virtuellen Realität bewegt sich der Benutzer beispielsweise in virtuellen 3D-Räumen oder trägt Hardware am Körper, welche ihm gleichzeitig virtuelle und reale Informationen zur Verfügung stellt. Um optimale Benutzerschnittstellen zur Verfügung zu stellen, fließen die Erkenntnisse von unterschiedlichen wissenschaftlichen Disziplinen wie Psychologie, Arbeitswissenschaft oder Design beim Entwurf und der Umsetzung der Benutzerschnittstellen mit ein. Mit der Zeit haben sich verschiedene Begriffe wie Benutzerfreundlichkeit, Bedienbarkeit, Gebrauchstauglichkeit oder Usability herauskristallisiert, welche die Qualität der Interaktion zwischen dem Menschen und dem Computer charakterisieren sollen. Eine hohe Usability von Software hat großen Einfluss auf die Produktivität der Nutzer. Sie kann z. B. Stress entgegenwirken, eine vorzeitige Ermüdung und ein Nachlassen der Konzentrationsfähigkeit verhindern. Konkrete Kriterien für eine ergonomische Gestaltung von Software sind in der ISO 9241 Teil 110 zu finden.



LE 2 - Grundlegende Anforderungen an Benutzerschnittstellen präsentieren

Die User Experience (UX) ist ein wesentliches Kriterium, welches bei der Akzeptanz eines Softwareproduktes durch den Kunden eine große Rolle spielt. Um eine gute UX zu erreichen sollte jedes Softwareprodukt einige grundlegende Anforderungen, wie z. B. die Usability oder Zugänglichkeit erfüllen.

User Experience

Unter User Experience werden die Nutzererfahrung bzw. die Wahrnehmungen und Reaktionen einer Person verstanden, die aus der tatsächlichen und der erwarteten Benutzung eines Produkts, eines Systems oder einer Dienstleistung resultieren.

Ein Verfahren, welches die UX bei der Entwicklung einer Benutzerschnittstelle in den Mittelpunkt stellt, ist das User Experience Design.

(1) User Experience Design

Das User Experience Design (UX Design) befasst sich mit der Analyse, Gestaltung und Optimierung der User Experience. Zum Beispiel wird versucht, komplexe Systeme einfach darzustellen und den Nutzer so bequem und schnell

wie möglich an das gewünschte Ziel zu bringen. Für ein gutes Softwaresystem ist gutes UX Design unabdingbar. Es gibt viele Beispiele für ein missglücktes UX Design, z.B. unübersichtliche Webseiten oder komplizierte Fahrkartenautomaten. Eine gelungene User Experience hingegen ist meistens nicht besonders auffällig.



Bestandteile des UX Designs	
Visuelles Design	Visuelles Design (UI-Design) beschreibt die Ästhetik des Softwareprodukts. Es stellt die abschließende Ebene des UX-Design-Prozesses dar. Dazu werden im UI-Design entsprechend Farben, Bilder und Symbole eingesetzt.
Informationsarchitektur	Die Informationsarchitektur umfasst das Strukturieren von Informationen, sodass der Nutzer sie möglichst intuitiv finden kann. Um Inhalte den Nutzern zugänglich zu machen, erstellen Informationsarchitekten entsprechende Navigationskonzepte.

Bestandteile des UX Designs	
Interaktionsdesign	Das Interaktionsdesign ist der wichtigste Teil des UX Design, welches das Ziel hat, Softwareprodukte zu entwickeln, die intuitiv und effizient nutzbar sind. Dazu befassen sich Interaktionsdesigner ausführlich mit der Nutzerforschung, um Erwartungen und Bedürfnisse ihrer Zielgruppen zu erfassen.
Usability (Gebrauchstauglichkeit)	Die Usability beschreibt, inwiefern ein Softwareprodukt den Nutzer dabei unterstützt, seine Ziele in einem bestimmten Nutzungskontext zu erreichen. Je einfacher die Bedienung ist, desto höher ist die Usability. Die Usability kann durch Usability-Tests objektiv gemessen werden. Aus diesen Tests können aber nur bedingt brauchbare Aussagen abgeleitet werden, da die Usability die subjektive Wahrnehmung des Nutzers nicht berücksichtigt.
Zugänglichkeit	Die Zugänglichkeit eines Softwaresystems beschreibt, wie leicht dessen Nutzung fällt und wie dessen Erreichbarkeit ist. Sie trägt dazu bei, die mit einem System verbundene Lernkurve zu verkürzen. Zugänglichkeit betrifft auch den Aufwand für Menschen mit Behinderungen eine Anwendung zu benutzen (siehe auch Barrierefreiheit).

(2) Corporate Design

Corporate Design ist ein Teilbereich der Corporate Identity. Unter Corporate Identity werden die gebündelten Merkmale und Charakteristika eines Unternehmens zusammengefasst, durch die es definiert wird. Durch Corporate Identity ist ein Unternehmen von anderen Unternehmen unterscheidbar und es soll nach innen und nach außen als eine Einheit dargestellt werden. Dies gelingt z. B. durch Grafiken wie Logos, Banner oder gestalterische Elemente, welche z. B. auf Briefköpfen, bei der Produktgestaltung oder bei Internetauftritten verwendet werden. Zur Corporate Identity gehören die drei grundlegenden Teilbereiche Corporate Culture (Unternehmenskultur), Corporate Communication (Unternehmenskommunikation) und das Corporate Design (Erscheinungsbild des Unternehmens), welches bei der Gestaltung von Benutzerschnittstellen den wichtigsten Bereich darstellt.



Corporate Design (CD)

Das Corporate Design umfasst alle Maßnahmen zur Entwicklung des visuellen Erscheinungsbilds eines Unternehmens. Ziel ist es, ein positives Bild des Unternehmens nach außen zu zeigen, einen hohen Wiedererkennungswert zu schaffen und die Kundenbindung zu erhöhen.

(3) Barrierefreiheit

Bei der Benutzbarkeit für den Endanwender sollten Anwendungsentwickler auch die Barrierefreiheit ihrer Software im Auge behalten. So können auch Personen mit körperlichen Einschränkungen die zu entwickelnde Software nutzen. Doch Barrierefreiheit ist mehr als nur die Entwicklung von Software für Menschen mit Körperbehinderung. Barrieren gibt es überall. Selbst Fremdsprachigkeit kann eine Barriere darstellen oder fortgeschrittenes Alter. Barrierefreiheit sollte dementsprechend grundsätzlich bei der Softwareentwicklung mitberücksichtigt werden. Der Aufwand dafür ist geringer als viele glauben. Alle wichtigen modernen Betriebssysteme bieten dazu globale Funktionen an, welche Entwicklerinnen und Entwickler nutzen können. Dazu

zählen beispielsweise die Bildschirmtastatur, die Unterstützung für Braille-Displays, kontrastreiche Darstellungen, Bildschirmlupen, das Vorlesen von Text oder die Spracheingabe von Befehlen.

Viele nützliche und weiterführende Informationen zur Barrierefreiheit sind auf der Webseite www.bundesfachstelle-barrierefreiheit.de/DE/Praxishilfen/Informationstechnik/informationstechnik_node.html [15.06.2021] zu finden.

Durch die Umsetzung der nachfolgenden Punkte kann bereits eine grundsätzliche Barrierefreiheit eines Programms gewährleistet werden. Der zusätzliche Entwicklungsaufwand ist gering, wenn die Software auf die Funktionalität der Barrierefreiheit in heutigen Betriebssystemen zurückgreift.

Beispiele für einfache Möglichkeiten zur Umsetzung der Barrierefreiheit

- Bedienelemente sollten mit der Tabulator-Taste auswählbar sein.
- Eine Skalierbarkeit der Inhalte ist zu gewährleisten, z. B. die Vergrößerung von Elementen.
- Die Plattformunabhängigkeit in Sachen Endgeräte und Displays ist zu gewährleisten, damit z. B. Braille-Displays jederzeit genutzt werden können.
- Grafiken sollten, wenn möglich, mit Alternativtexten versehen werden. Ein Button zum Speichern kann z. B. per Schnellinformation (Tooltip) und Alternativtext auch problemlos von Nutzern mit Sehbehinderung verwendet werden.

Sind weitere spezielle Umsetzungen von Barrierefreiheit laut Kundenauftrag erforderlich, sollten diese genauestens mit dem Kunden abgesprochen werden, da ihre Entwicklung einen nicht unerheblichen Kostenfaktor darstellen kann.

LE 3 - Eine Oberfläche für eine Benutzerschnittstelle entwerfen

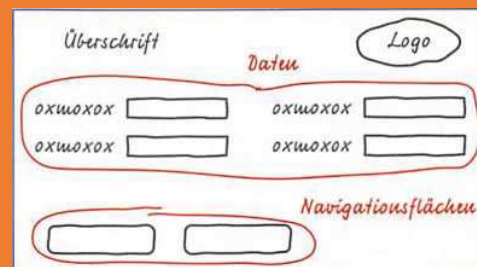
Bevor mit der Umsetzung eines User Interfaces begonnen wird, sollten verschiedene grafische Entwürfe von der Oberfläche angefertigt werden. Diese Entwürfe sollen die Entwicklerinnen und Entwickler bei der Gestaltung der Benutzerschnittstelle und der Planung einer optimalen Benutzerführung unterstützen. Auf der anderen Seite werden solche Entwürfe auch bei Kundengesprächen genutzt, um erste Entwürfe des UX Design präsentieren zu können. In der Regel durchläuft der Entwurf einer GUI in folgenden Stufen:



Entwürfe von der Oberfläche

Sketch

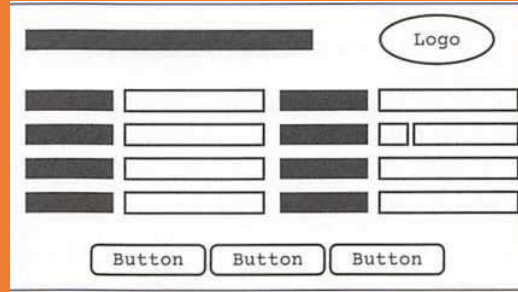
Ein Sketch (Skizze) ist der erste Entwurf im Designprozess. Im Unterschied zu einem Wireframe sind Skizzen nur grob gehalten und unsauber. Dabei geht es weniger um Ästhetik und Perfektion als vielmehr um das schnelle konzeptionelle Festhalten von Ideen. Es geht hier ausschließlich um das Verständnis, was der Benutzer sieht. Sketches kann man mit Stift und Papier, auf einem Whiteboard oder auch digital erstellen.



Beispiel für eine Skizze von einem Dialog zur Erfassung der Daten von Vereinsmitgliedern

Wireframes

Ein Wireframe ist ein sehr einfacher, konzeptioneller Entwurf einer Applikation oder Internetseite. Wireframes lassen sich in statische und dynamische Wireframes einteilen. Unter einem statischen Wireframe versteht man eine schematische Darstellung von nur einer einzelnen Oberfläche. Für diese werden die grundlegenden Elemente skizziert. Bei dynamischen Wireframes werden mehrere Oberflächen miteinander verknüpft. Man spricht in diesem Zusammenhang von dem funktionellen Entwurf.



Beispiel für ein Wireframe von einem Dialog zur Erfassung der Daten von Vereinsmitgliedern

Wesentliche Merkmale:

- keine Farben (nur schwarz, weiß, grau)
- keine Typographie
- keine Bilder (nur Platzhalter)
- keine Texte
- statische Darstellungsform
- Seitenstruktur
- Augenmerk auf Anordnung, Größe und Platzierung der Elemente

Wireframes werden z. B. zur Dokumentation eines Projekts oder zur Kommunikation innerhalb des Entwicklerteams eingesetzt. Sie dienen als Vorlage für das Mockup.

Tools: Es gibt viele Möglichkeiten, Wireframes mithilfe von Tools zu erstellen, z. B. Balsamiq, Axure, UXPin, Adobe XD oder Omnigraffle. Aber auch Papier und Bleistift sind noch üblich.

Beispiele für Wireframes:



Mockup

Ein Mockup wird auf der Grundlage eines Wireframes erstellt und ist weitaus detaillierter. Hier werden z. B. die Inhaltselemente mit Farben versehen und die Schriftart und -größe festgelegt.

Das Ziel ist es, so genau wie möglich das spätere Erscheinungsbild der Applikation oder Internetseite darzustellen und dem Kunden einen exakten Entwurf zu präsentieren. Allerdings stellt das Mockup noch kein fertiges Design des Produktes dar. Das Mockup wird später in einen Prototyp überführt.

Beispiel für ein Mockup von einem Dialog zur Erfassung der Daten von Vereinsmitgliedern

Wesentliche Merkmale:

- „look & feel“ der Anwendung
- Farben
- Typographie
- Bilder
- Blindtexte
- Modell mit rudimentärer Funktionalität

Tools: Im einfachsten Fall kann man ein Mockup mit Papier und Bleistift erstellen. In der Regel wird aber ein Mockup mit Tools wie Adobe Photoshop, Adobe XD, Balsamiq, Axure oder UXPin erstellt.

Beispiele für Mockups



Prototyp

Ein Prototyp stellt schon eine funktionstüchtige Anwendung dar, bei der man bereits Interaktionen durchführen kann. Das heißt, man kann beispielsweise die Navigation testen und sich ein sehr genaues Bild des Zusammenspiels zwischen Design und Inhalt machen. Ein Prototyp ist aber immer noch kein fertiges Produkt und wird im Laufe der Entwicklungszeit immer mehr optimiert.

TAG 10 - Vertiefung des Wissens und Modulprüfung Teil 1

LE 1 - Einleitung

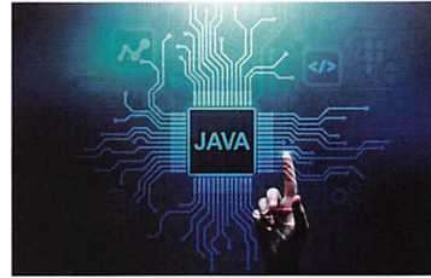
TAG 10 in diesem Modul bietet Ihnen die Möglichkeit der Wiederholung und Vertiefung Ihres Wissens. Sollten es noch offene, nicht vollständig bearbeitete Modul-Inhalte geben, nutzen Sie den heutigen Tag zur Komplettierung.

LE 2 – Modulprüfung, Teil 1

120 Minuten zu denen wir Ihnen recht viel Erfolg wünschen!

TAG 11 - TAG 13 - Anwendungen in Java implementieren

Java gehört zu den populärsten Programmiersprachen der Welt. In dem TIOBE-Index stand Java entweder auf Platz 1 oder Platz 2 in der Liste und wird für die unterschiedlichsten Anwendungen eingesetzt. Deswegen sollte man zumindest grundlegende Kenntnisse in der Entwicklung von Java-Programmen besitzen. Dieses Kapitel gibt dementsprechend einen kurzen und kompakten Einstieg in die Java-Programmierung. Dabei werden grundlegende Programmierkenntnisse, welche im ersten Lehrjahr erworben wurden, vorausgesetzt und auf diesen aufgebaut. Betrachtet werden u.a. grundlegende Sprachelemente, z. B. Schlüsselwörter und Kontrollstrukturen, die Entwicklung von Konsolen- und GUI-Anwendungen und die Umsetzung des objektorientierten Programmierparadigmas in Java.



LE 1 - Java beschreiben und eine Entwicklungsumgebung auswählen

Java ist eine moderne, populäre und einfach zu erlernende Programmiersprache. In der Praxis gehört Java zu den am meisten eingesetzten Programmiersprachen und wird für alle möglichen Anwendungsfälle verwendet. Deswegen gibt es auch entsprechend viele Entwicklungsumgebungen, mit deren Hilfe Java- Programme erstellt werden können.

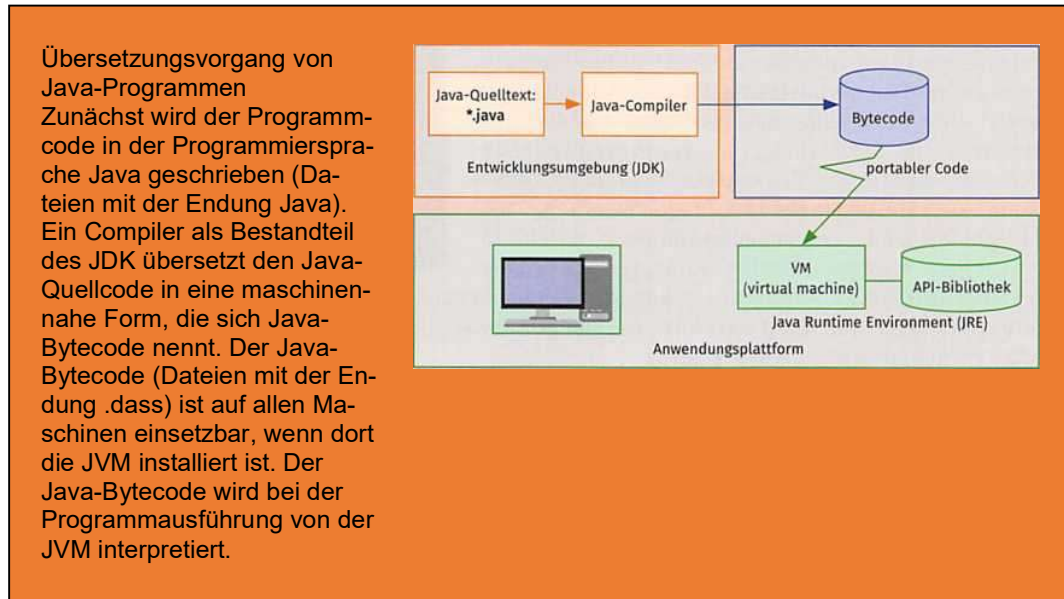
Die Programmiersprache Java und geeignete Entwicklungsumgebungen

Java ist eine von der Firma SUN entwickelte Sprache und seit 1995 auf dem Markt. Sie ist immer noch eine junge Programmiersprache und wird dadurch noch vielen Wandlungen unterworfen sein. Im Jahre 2010 wurde SUN durch die Firma Oracle übernommen und seitdem wird Java von Oracle weiterentwickelt.

Java steht für eine spezielle Technologie der Bereitstellung von Software und hat sich quasi zu einer Philosophie der IT-Branche entwickelt. Als Programmiersprache unterstützt Java den Programmierer dabei, kleine, zuverlässige und fehlerfreie Programmeinheiten (Module) zu erstellen. Wichtig ist in diesem Zusammenhang die konsequente Objektorientierung. Unter Java entstehen Anwendungen mit gesicherter Qualität, wobei die Performance etwas in den Hintergrund tritt. Als Technologie umfasst Java zum einen das JDK (Java Development Kit) als komfortable Entwicklungsumgebung und zum anderen die JRE (Java Runtime Environment) als Laufzeitumgebung. In der Entwicklungsumgebung JDK werden die Programme erfasst, getestet und übersetzt. Die Laufzeitumgebung realisiert die Java-Plattform und muss auf den Computern installiert sein, auf denen die Java-Anwendungen ausgeführt werden sollen. Die Laufzeitumgebung beinhaltet zum einen die virtuelle Maschine (VM) zur Interpretation und Ausführung des vorab übersetzten Java-Programms und zum anderen die Java-API (Application Programming Interface) als Bibliothek mit vorgefertigten Softwarebausteinen. Java zielt als Philosophie auf Unabhängigkeit und Qualität. Unabhängig will man von konkreten Hard- und Softwareplattformen sein, unabhängig aber auch von jeglichen kommerziellen Einflüssen. Java ist das Werkzeug der Open-Source-Community.

Um Java-Programme auszuführen, muss die JVM auf dem Rechner installiert sein. Die Vorgehensweise für die Installation von Java für bestimmte Betriebssysteme ist auf der Seite www.java.com/de/download/help/download_options.html [15.06.2021] zu finden. Das JDK kann auf der Seite www.oracle.com/de/java/technologies/javase-downloads.html [15.06.2021] heruntergeladen werden.

In Java geschriebene Programme werden in einem zweifachen Übersetzungsvorgang für die Maschine lesbar und ausführbar gemacht. Dieser wird hier kurz näher erläutert.



In der Regel wird für die Programmentwicklung mit Java eine Entwicklungsumgebung (IDE) verwendet. Es gibt eine Reihe IDE, welche sich hervorragend für diese Aufgabe eignen. Dabei gibt es unterschiedliche Gründe, sich für die eine oder andere Variante zu entscheiden. Einige wichtige IDE sind zum Vergleich in der folgenden Tabelle aufgeführt.

Entwicklungsumgebungen für Java	
Eclipse	Eclipse ist eine frei verfügbare IDE und ist in der Java Community sehr verbreitet. Es gibt Versionen für Windows, Linux und MacOS und die Entwicklungsumgebung kann durch viele Plug-Ins erweitert werden, z. B. einen GUI-Builder.
IntelliJ	IntelliJ ist eine moderne und leistungsfähige Entwicklungsumgebung des Unternehmens JetBrains für die Programmiersprache Java. Die IDE läuft unter Windows, Linux und MacOS. Es gibt eine kommerzielle und eine frei nutzbare Version. Die freie, quelloffene Community Edition deckt alles ab, was zur Java SE-Entwicklung nötig ist. Die kommerzielle Ultimate Edition richtet sich an Java EE-Entwickler.
NetBeans	NetBeans ist eine freie IDE, welche komplett in Java geschrieben wurde und damit auf allen Plattformen läuft, auf denen die JVM installiert ist. Sie bietet komfortable Möglichkeiten zur Java SE- und Java EE-Entwicklung.
BlueJ	BlueJ ist eine freie IDE, die speziell für Ausbildungszwecke konzipiert wurde. BlueJ ist in Java geschrieben und läuft damit auf allen gängigen Plattformen wie Windows, Linux oder MacOS.
VSCode	Ist ein freier Quelltext-Editor von Microsoft, welcher durch Extensions für die Java- Programmierung erweitert werden kann. Läuft unter Windows, Linux und MacOS.

Da Eclipse eine der am häufigsten eingesetzten IDE für die Entwicklung von Java-Programmen ist und durch die Plug-In-Schnittstelle auch ein GUI-Builder installiert werden kann, wird diese im Buch verwendet und im Folgenden näher betrachtet.

Entwicklungsumgebung Eclipse

Eclipse ist eine sehr flexible, leistungsfähige und quelloffene Entwicklungsumgebung, die nach entsprechender Aufrüstung durch Plug-ins durchaus mit kommerziellen IDEs konkurrieren kann. Eclipse selbst basiert auf Java und wurde ursprünglich für die Entwicklung von Java-Programmen entwickelt. Mittlerweile wird Eclipse auch für viele andere Aufgaben im Bereich der Softwareentwicklung eingesetzt. Der Grund dafür ist, dass heutzutage eine Vielzahl von quelloffenen und kommerziellen Erweiterungen für Sprachen wie HTML, CSS oder PHP erhältlich sind. Eclipse kann schnell und einfach installiert werden und läuft sowohl unter Windows, MacOS oder Linux. Deswegen wird diese IDE von sehr vielen Java-Entwicklern genutzt.

Die aktuelle Version von Eclipse kann unter www.eclipse.org/downloads [15.06.2021] heruntergeladen und danach installiert werden.

LE 2 - Grundlegende Sprachelemente beschreiben und Konsolenanwendungen implementieren

In diesem Abschnitt wird eine kurze Einführung in die grundlegenden Sprachelemente von Java gegeben und es werden die Grundlagen gelegt, um mit der Entwicklungsumgebung Eclipse einfache Konsolenanwendungen zu erstellen. Entwicklern mit Erfahrungen in Sprachen wie C++ oder C# wird die Syntax von Java sehr bekannt vorkommen.



Java Programme mithilfe von Eclipse erstellen und ausführen

Es gibt die Möglichkeit, in Eclipse verschiedenste Einstellungen für Java-Projekte vorzunehmen und diese an die eigenen Bedürfnisse anzupassen. Nachfolgend wird ein einfacher Weg beschrieben, um ein Java-Projekt zu erstellen. So kann man schnell und unkompliziert seine Konsolenanwendungen ausprobieren.

Java-Programm mithilfe von Eclipse erstellen

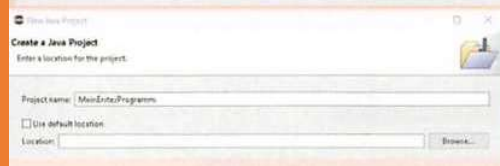
Programm starten und den Workspace eingeben

Gestartet wird Eclipse durch einen Doppelklick auf das Programmsymbol. In das folgende Fenster muss dann das Arbeitsverzeichnis eingegeben und mit „Launch“ bestätigt werden.



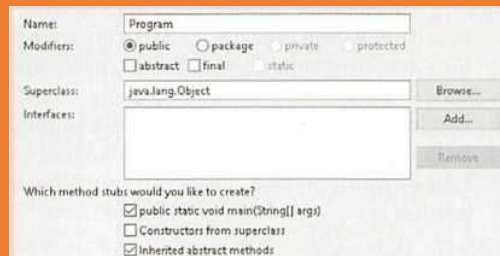
Projekt anlegen

Durch den Menübefehl „File | New | Java Project“ wird ein neues Projekt erzeugt. Es ist dabei ein Projektname anzugeben, hier z. B. „MeinErstesProgramm“, und mit Betätigen des „Finish“-Buttons zu bestätigen.



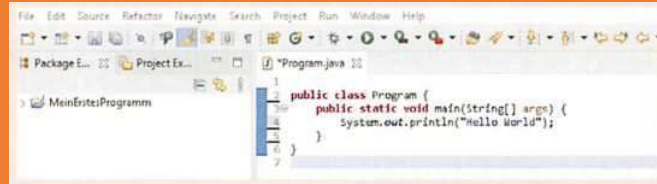
Projektdatei erzeugen

Durch den folgenden Menübefehl wird eine neue Projektdatei mit einer Klasse erzeugt: „File | New | Class“. Dabei ist ein Name für die Klasse anzugeben, hier z. B. der Name „Program“. Weiterhin ist der Haken bei dem Auswahlpunkt „public static void main (String[] args)“ zu setzen. Danach wird durch das Betätigen des Buttons „Finish“ eine entsprechende Klasse erzeugt.



Programm schreiben

Jetzt sollte die abgebildete Ansicht zu sehen sein und es kann mit dem Schreiben des Programmes begonnen werden. Als erstes Programm soll das klassische „Hello World“ ausprobiert werden.



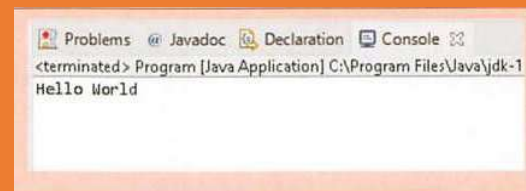
Nachfolgend ist der Quelltext für dieses Programm angegeben.

```
public class Program {
    public static void main( String[] args ) {
        System.out.println("Hello World");
    }
}
```

Dieser Quelltext kann in die IDE „Eclipse“ zum Ausprobieren übertragen werden. Dabei ist zu beachten, dass jene Teile, die automatisch generiert wurden und im Quelltext schon vorhanden sind, nicht erneut eingegeben werden müssen. In unserem Fall reicht es also, die Anweisung „System.out.println(„Hello World“);“ zu übertragen.

Programm starten

Durch die Menübefehle „Run | Run“ oder „Strg + F11“ kann das Programm gestartet werden. Das Programm wird ausgeführt und das Ergebnis erscheint im Ausgabefenster von Eclipse.



LE 3 - Grundlegende Sprachelemente

Nachfolgend wird ein kurzer Überblick über grundlegende Sprachelemente in Java gegeben. Wie schon erwähnt ist der Überblick sehr kompakt gehalten und es werden grundlegende Kenntnisse in der Programmierung vorausgesetzt. Es werden u.a. die wichtigsten Operatoren, Kontrollfunktionen Felder und Zufallszahlen betrachtet. Die Beispiele in diesem Abschnitt sind zunächst einmal als Konsolen-Programme realisiert.

Groß- und Kleinschreibung, Semikolons und geschweifte Klammern

Java ist „case sensitive“, d. h., es wird zwischen Groß- und Kleinschreibung unterschieden. So sind in Java „auto“ und „Auto“ zwei unterschiedliche Bezeichner. Hier schleichen sich, besonders am Anfang, oft Syntaxfehler ein.

Eine andere Fehlerquelle am Anfang ist, dass das Setzen von Semikolons vergessen wird. Diese kennzeichnen das Ende einer Anweisung. Wird ein Semikolon vergessen oder falsch gesetzt, wird dieses als Syntaxfehler gekennzeichnet. Auf der anderen Seite dürfen aber mehrere Semikolons hintereinanderstehen, z. B. Anweisung;;;;;. Das ergibt aus programmtechnischer Sicht aber keinen Sinn.

Beispiele für das Setzen von Semikolons

```
</> a = a + 10;
      System.out.print("Hallo Welt");
      float summe = 0;
```

Eine mögliche dritte Quelle für Syntaxfehler besteht im Vergessen oder falschen Setzen von geschweiften Klammern. Diese fassen Codeabschnitte in zusammengehörige Blöcke zusammen. Man spricht in diesem Zusammenhang auch von Anweisungsblöcken. Ein Anweisungsblock beginnt mit einer öffnenden Klammer und wird mit einer schließenden Klammer beendet. Jeder Anweisungsblock kann eine beliebige Anzahl von Anweisungen enthalten. Blöcke können beliebig ineinander verschachtelt werden.

Beispiel für das Setzen von geschweiften Klammern

```
</> public class BeispielKlammern {    // öffnende Klammer (Klasse)
    public static void main( String[] args ) {    // öffnende Klammer (Methode)
        a = 0;
        while ( a < 10 ) {    // öffnende Klammer (Anweisungsblock)
            .....
        }    // schließende Klammer (Anweisungsblock)
    }    // schließende Klammer (Methode)
}    // schließende Klammer (Klasse)
```

Bezeichner und Literale

Mithilfe von Bezeichnern wird beispielsweise eine Variable, eine Klasse oder eine Methode eindeutig identifiziert. Man spricht in diesem Zusammenhang auch von Variablen-, Klassen- oder Methodennamen. Diese Namen müssen immer eindeutig sein und können im Rahmen nachstehender Regeln frei gewählt werden.

Regeln für Bezeichner in Java

- Ein Bezeichner darf nur alphanumerische Zeichen und Unterstriche enthalten.
- Leerzeichen und andere Sonderzeichen, wie #, \$ oder % sind nicht zugelassen.
- Ein einzelner Unterstrich allein ist nicht zulässig.
- Ein Bezeichner muss mit einem Buchstaben oder dem Unterstrich anfangen.

Beispiele: Raumfahrzeug, umsatz_, _vorname, anzahlPersonen, x_Abstand, y1, y2

Schlüsselwörter

Bei den Schlüsselwörtern handelt es sich um reservierte Wörter, die eine besondere Bedeutung in Java besitzen. Sie können nur im vorgesehenen Zusammenhang verwendet werden und sind als Bezeichner (z. B. für Variablen) nicht erlaubt.

Schlüsselwörter in Java				
abstract	continue	for	new	switch
assert	default	if	package	synchronized
boolean	do	goto	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Kommentare

Mithilfe von Kommentaren können Notizen und Bemerkungen direkt in den Quellcode mit aufgenommen werden. Folgende Kennzeichnungen für Kommentare sind möglich.

Kommentar in Java		
//	einzeilige Kommentare, reichen bis zum Ende der Zeile	// Dies ist ein Kommentar
/*... */	mehrzeilige Kommentare, reichen über mehrere Zeilen	/* Dies ist ein Kommentar, welcher über mehrere Zeilen geht */

Datentypen

Der Datentyp einer Variablen oder Konstanten legt den Wertebereich sowie die Größe des Speicherplatzes fest. Des Weiteren bestimmt er die Operationen und Funktionen, die für diesen Datentyp angewendet werden können (Grundlagen wie digitale Abbildung von Zahlen und Text.

Datentypen in Java			
Kategorie	Datentyp	Wertebereich	Speichergroße
Ganzzahl-Datentypen Ganzzahl-Datentypen besitzen keine Nachkommastellen und werden im Computer immer genau dargestellt.	byte	-128 ... + 127	1 Byte
	short	-32768 ... +32767	2 Byte
	int	-2.147.483.648... +2.147.483.647	4 Byte
	long	-9.223.372.036.854.775.808 ... +9.223.372.036.854.775.807	8 Byte
Gleitkomma-Datentypen Gleitkomma-Datentypen enthalten Kommastellen. Der Computer kann jedoch nicht jede Zahl genau darstellen. Je nach verwendetem Typ ist eine bestimmte Genauigkeit in den Nachkommastellen möglich.	float	1.4*10 ⁻⁴⁵ ... 3.4028235 *10 + 38	4 Byte
	double	5 *10 - 324... 1.797693134 *10 + 308	8 Byte
Zeichen-Datentypen Der Zeichen-Datentyp „char“ in Java kann beliebige Zeichen des Unicode (siehe auch Jahrgangsband 1, Lernfeld 5) enthalten. Einzelne Zeichen werden bei der Zuweisung durch Apostrophe eingeschlossen.	char	0 ... 65536 (Unicode-Zeichen von \u0000 ... \uffff)	2 Byte
Boolescher Datentyp Dieser Datentyp kann als Wert nur die Literale „true“ oder „false“ annehmen. Logische Ausdrücke liefern in Java immer einen booleschen Wert zurück.	boolean	true, false	1 Byte

Variablen und Konstanten

Variablen können während der Programmausführung unterschiedliche, veränderbare Werte annehmen, z.B. Zwischen- und Endergebnisse aus Berechnungen. Für jede Variable wird ein Speicherplatz im Arbeitsspeicher des Computers reserviert, auf den mit dem Variablennamen zugegriffen werden kann. Variablen werden mit einem Namen und einem Datentyp vereinbart.

Jede Variable, die in einem Programm verwendet wird, muss vor ihrer ersten Verwendung deklariert werden.

Beispiele für die Deklaration von Variablen

```
</> // Deklaration einer einzelnen Variablen
int anzahl;
// Deklaration einer einzelnen Variablen und gleichzeitige Initialisierung
double preis = 12.30;
// Deklaration mehrerer Variablen
double breite, laenge, hoehe;
```

Konstanten können wie Variablen deklariert werden und Werte aufnehmen. Im Gegensatz zu Variablen kann der vereinbarte Wert aber während des Programmablaufs nicht mehr verändert werden. Wird dies trotzdem versucht, reagiert das Programm mit einer Fehlermeldung. Konstanten werden in Java mit dem Schlüsselwort „*final*“ gekennzeichnet und für die Bezeichner werden üblicherweise nur Großbuchstaben verwendet.

```
</> final double PI = 3.1415;
final int ANZAHL_MONATE = 12;
final byte BLAU = 234;
```

Operatoren

Operatoren sind Symbole, die Verarbeitungsaktionen in Ausdrücken auslösen. In einem Ausdruck werden die Operatoren hinsichtlich der Ausführungsreihenfolge nach Prioritäten eingeteilt. Die Ausführungsreihenfolge kann auch durch das Setzen von runden Klammern festgelegt werden. Der Inhalt der Klammern wird immer zuerst ausgewertet, da ihm die höchste Priorität zukommt. Eine Auswahl wichtiger Operatoren ist in der nachfolgenden Tabelle aufgeführt.

Operatoren in Java			
Art	Operator	Bezeichnung	Bedeutung
Zuweisungsoperator	=	Einfache Zuweisung	Speicherung eines Wertes in einem Objekt
Arithmetische Operatoren	+	Addition	a + b ergibt die Summe von a und b
	-	Subtraktion	a - b ergibt die Differenz von a und b
	*	Multiplikation	a * b ist das Produkt aus a und b
	/	Division	a / b ist der Quotient von a und b
	%	Modulo	a % b ist der Rest der ganzzahligen Division von a und b
Relationale Operatoren	=	gleich	a = b ergibt true, wenn a und b gleich sind
	!=	ungleich	a != b ergibt true, wenn a und b ungleich sind
	<	kleiner	a < b ergibt true, wenn a kleiner b ist
	>	größer	a > b ergibt true, wenn a größer b ist
	<=	kleiner gleich	a <= b ergibt true, wenn a kleiner oder gleich b ist
	>=	größer gleich	a >= b ergibt true, wenn a größer oder gleich b ist

Operatoren in Java			
Art	Operator	Bezeichnung	Bedeutung
Boolesche Operatoren	!	Negation	invertiert den Ausdruck
	&&	und	ergibt true, wenn beide Operanten true sind
		oder	ergibt true, wenn mindestens ein Operant true ist

LE 4 - Ein- und Ausgabe in der Konsole

Zur Ausgabe von Zeichenketten, Zahlen und weiteren Elementen verwendet man *System.out.print* oder *System.out.println*. Für die Ausgabe werden Zeichenketten in Anführungsstriche eingeschlossen. Variablen werden mit Zeichenketten durch ein „+“ verbunden.

Beispiele für die Konsolenausgabe in Java

```
</> // einfache Ausgabe ohne Zeilenwechsel
System.out.print ("Geben Sie einen Wert ein: ");
// Ausgabe mit Zeilenwechsel
System.out.println ("Geben Sie einen Wert ein: ");
// Ausgabe Text + Variable
System.out.println ("Das Ergebnis ist: " + ergebnis);
```

Die Konsoleneingabe untergliedert sich in folgende Schritte:

1. Einbinden der Import-Anweisung für das Paket „java.util“, da dort die Scanner-Klasse definiert ist.

```
</> import java.util.*;
```

2. Anlegen eines Objektes der Klasse „Scanner“, welche als Eingabestrom die Standardeingabe „System.in“ nutzt. Es genügt, sich zu merken, dass der Eingabepuffer benötigt wird. Mit dem Deklarieren und Anlegen von Speicherplatz für den Scanner besteht nun die Möglichkeit, direkt Eingaben von der Eingabekonzole vorzunehmen.

```
</> Scanner input = new Scanner(System.in)
```

3. Deklaration einer Variablen, in die der Inhalt von Scanner geschrieben wird, sowie Einlesen der Konsoleneingabe des Benutzers.

```
</> String text = input.nextLine();
```

Beim Einlesen von Integer-Zahlen empfiehlt es sich, anstatt von „nextLine()“ die Methode „nextInt()“ und für Kommazahlen die Methode „nextDouble()“ zu verwenden.

4. Am Ende muss der Eingabestrom wieder geschlossen werden. Dieses wird durch die Methode „close()“ erreicht.

```
</> input.close();
```

Die Konsoleneingabe und -ausgabe in Java wird in dem nachfolgenden Beispiel noch einmal verdeutlicht. In diesem wird der Benutzer aufgefordert, verschiedene Daten einzugeben, und im Anschluss daran werden die eingegebenen Werte wieder ausgegeben.

Beispiel für die Konsoleneingabe und -ausgabe in Java

```
</> import java.util.*;
public class BeispielEinUndAusgabe {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Geben Sie bitte ihren Namen ein");
        String name = input.nextLine();
        System.out.println("Geben Sie bitte ihr Alter ein");
        int alter = input.nextInt();
        System.out.println("Geben Sie bitte Ihre Körpergröße ein");
        double groesse = input.nextDouble();
        System.out.println("Sie heißen " + name + ", sind " + alter +
                           "Jahre alt und sind " + groesse + " groß.");
        input.close();
    }
}
```

LE 5 - Kontrollstrukturen

Kontrollstrukturen dienen dazu, den Programmablauf zu kontrollieren. Damit bietet eine Kontrollstruktur die Möglichkeit, Programmteile nur unter gewissen Bedingungen auszuführen oder Programmteile zu wiederholen. Auch in Java findet man alle Kontrollstrukturen, welche aus anderen Programmiersprachen bekannt sind. Programmierern, welche schon Erfahrungen in C++ oder C# oder ähnlichen Sprachen mitbringen, wird die Syntax sehr vertraut vorkommen.

Verzweigungen

Die Verzweigung (Auswahl, Alternative) unterstützt den Programmablauf, wenn er von einer oder mehreren Bedingungen abhängt.

Verzweigungen in Java		
Art	Syntax	Beispiele
Einseitige Verzweigung	if (Bedingung) { Anweisung1; Anweisung2;	<pre>Scanner input = new Scanner(System.in); System.out.println("Eingabe Zahl: "); double zahl = input.nextDouble(); if (zahl > 0) { System.out.println("Die Zahl " + zahl + " ist größer als Null"); }</pre>
Zweiseitige Verzweigung	<pre>if (Bedingung) { Anweisung11; Anweisung12; } else { Anweisung21; Anweisung22;</pre>	<pre>Scanner input = new Scanner(System.in); System.out.println("Eingabe Zahl: "); double zahl = input.nextDouble(); if (zahl >= 10 && zahl <= 20) { System.out.println("Die Zahl " + zahl + " liegt im Bereich von 10 bis 20"); } else { System.out.println("Die Zahl " + zahl + " liegt nicht im Bereich von 10 bis 20"); }</pre>

Verzweigungen in Java		
Art	Syntax	Beispiele
Fallunter-scheidung	<pre>switch (Selektor) { case Wert1: ... break; case Wert2: ... break; default: ... break; }</pre>	<pre>Scanner input = new Scanner(System.in); int auswahl = input.nextInt(); switch (auswahl) { case 1: Aufgabel(); // Funktionsaufruf break; case 2: Aufgabe2(); // Funktionsaufruf break; default: System.out.println("Das Programm " + "wird beendet."); break; }</pre>

Schleifen

Mithilfe von Schleifen können bestimmte Anweisungen mehrfach wiederholt werden. Schleifen bestehen aus einer Schleifensteuerung und einem Schleifenkörper. Es werden in Java grundsätzlich drei Arten von Schleifen unterschieden: kopfgesteuerte Schleifen, fußgesteuerte Schleifen und Zählschleifen.

Schleife in Java		
Art	Syntax	Beispiel
Kopfgesteuerte Schleife	<pre>while (Bedingung) { Anwei- sung1; Anweisung2; ... }</pre>	<pre>int i = 1; while (i <= 10) { System.out.println(i); i = i + 1; }</pre>
Fußgesteuerte Schleife	<pre>do { Anweisung1; Anweisung2; ... } while (Bedingung);</pre>	<pre>int i = 1; do { System.out.println(i); i = i + 1; } while (i <= 10);</pre>
Zählschleife	<pre>for (Startwert; Bedingung; Schrittweite) { Anweisung1; Anweisung2; ... }</pre>	<pre>for(int i = 1; i <= 10; i++) { System.out.println(i); }</pre>

LE 6 - Funktionen

In der nachfolgenden Übersicht ist der grundlegende Aufbau einer Funktion in Java zu erkennen. Dabei werden Funktionen mit und ohne Übergabeparameter und Rückgabewert unterschieden. Die einzelnen Bestandteile werden an dem nachfolgenden Beispiel erklärt.

```
</> double berechneSumme (double x, double y) {
  double summe = x + y;
  return summe;
}
```

Aufbau einer Funktion in Java	
<code>double</code>	Datentyp des Rückgabewertes der Funktion: Gibt die Funktion keinen Wert zurück, steht an dieser Stelle das Schlüsselwort „ <i>void</i> “.
<code>berechneSumme</code>	Name der Funktion, der frei gewählt werden kann. Es sollte aber ein aussagekräftiger Name sein, aus dem man die Aufgabe der Funktion ableiten kann.
<code>(double x, double y)</code>	Liste von Übergabeparametern an die Funktion: Diese Liste kann auch leer sein und besteht in diesem Fall nur aus den beiden Klammern.
<code>{</code>	Öffnende Klammer des Funktionsrumpfes.
<code>double summe = x + y;</code>	Einzelne Anweisungen, die in der Funktion ausgeführt werden.
<code>return summe;</code>	Rückgabewert der Funktion: Diese Anweisung ist nur erforderlich, wenn die Funktion einen Rückgabewert haben soll.
<code>}</code>	Schließende Klammer des Funktionsrumpfes.

Ein komplettes Programm mit einer Funktion und dem Aufruf derselben in der main-Funktion ist in dem nachfolgenden Quelltext zu sehen.

Beispiel einer Funktion und dem Aufruf derselben in der main-Funktion

```

</> import java.util.*;
public class Program {
    //-----
    // Start - Funktion eines jeden Java-Programms
    //-----
    public static void main(String[] args) {
        double summe = berechneSumme(17.3, 18.4); // Funktionsaufruf
        System.out.println(summe);
    }

    //-----
    // Anlegen der Funktion berechneSumme
    //-----
    static double berechneSumme (double x, double y) {
        double summe = x + y;
        return summe;
    }
}

```

LE 7 - Felder und Zufallszahlen

Felder

Die bisher verwendeten Variablen können immer genau einen Wert des entsprechenden Datentyps speichern. Um mehrere Werte des gleichen Datentyps zu speichern, werden Felder verwendet. In Java versteht man unter einem Feld (Array) einen Container, in dem mehrere Objekte vom gleichen Typ aufgenommen und verwaltet werden. Auf die einzelnen Feldelemente des Feldes kann man über Indizes zugreifen. Dabei beginnt jedes Feld mit dem Index 0.

Beispiele für das Anlegen und den Zugriff auf Felder

```
</> int[] feld = new int[5]; // Anlegen eines Feldes für fünf Integer-Werte
    feld[0] = 20;           // dem ersten Feldelement den Wert 20 zuweisen
    feld[4] = 10;           // dem letzten Feldelement den Wert 10 zuweisen

    // Anlegen eines Feldes für die monatlichen Umsätze eines Jahres
    double[] umsatz = new double [12];
    // Die Umsätze der Monate Januar und Februar addieren
    double summe = umsatz[0] + umsatz[1];

    // Anlegen eines zweidimensionalen Arrays mit 3 Zeilen und 5 Spalten
    double feld2D[][] = new double[3][5];
    // dem Feldelement in der Zeile 0 und der Spalte 0 den Wert 1 zuweisen
    feld2D[0][0] = 1;
    // dem Feldelement in der Zeile 2 und der Spalte 1 den Wert 7 zuweisen
    feld2D[2][1] = 7;
```

Zufallszahlen

Eine einfache Möglichkeit zur Erzeugung von Zufallszahlen in Java bietet die Klasse „Math“. Die Methode „Math.random()“ erzeugt eine Zufallszahl im Bereich größer gleich 0.0 und kleiner 1.0. Um andere Zufallszahlen zu erhalten, muss die erzeugte Zufallszahl noch durch weitere mathematische Operationen angepasst werden. Dieses Vorgehen wird anhand eines Beispiels nochmals aufgezeigt.

Beispiel für das Füllen eines Feldes mit Zufallszahlen

```
</> import java.util.*;
    public class BeispielZufallszahlen {
        public static void main(String[] args) {
            int anzahlElemente = 20;
            int[] feld = new int[anzahlElemente];
            for(int i = 0; i < anzahlElemente; i++) {
                feld[i] = (int)(Math.random() * 10); // Erzeugen der Zufallszahlen
                System.out.println(feld[i]);
            }
        }
    }
```

LE 8 - Exception Handling

Das Exception Handling in Java unterteilt sich in drei Blöcke, welche alle eine spezielle Funktion erfüllen.

Block	Beschreibung
try	Hier steht der Quelltext, in dem eine Exception auftreten könnte.
catch	In diesen Block wird gewechselt, wenn im try-Block eine Exception auftritt. Die Exception kann hier behandelt werden, d. h., der Fehler kann ausgewertet und entsprechend reagiert werden.
finally	Dieser Block wird immer ausgeführt, egal ob im try-Block eine Exception auftritt oder nicht. Tritt eine Exception auf, wird zuerst der catch-Block abgearbeitet und danach der finally-Block.

Die Anwendung des Exceptions-Handlings in Java wird nochmals in einem Beispiel verdeutlicht.

Beispiel für Exception-Handling in Java

```
</> import java.util.*;
public class BeispielExceptionHandling {
    public static void main(String[] args) {
        try {
            Scanner input = new Scanner(System.in);
            System.out.println("Geben Sie bitte eine ganze Zahl ein");
            int zahl = input.nextInt();
            System.out.println("Sie haben eine " + zahl + "eingegeben");
        } catch (Exception ex) {
            System.out.println("Es ist ein Fehler aufgetreten " + ex);
        } finally {
            System.out.println("Das Programm wird beendet");
        }
    }
}
```

Weitere Beispielaufgabe

Die Arbeit mit der Konsole in Java soll nun anhand einer Beispielaufgabe nochmals dargestellt werden.

Beispielaufgabe

Es soll eine Anwendung in Java entwickelt werden, in welche die Noten einer Klassenarbeit eingegeben werden können. Zusätzlich soll sie im Anschluss die Anzahl der einzelnen Noten und den Notendurchschnitt ermitteln und ausgeben.

Lösung

Für diese Aufgabe gibt es mehrere Lösungen. In diesem Beispiel werden die Noten und deren Anzahl in einem Feld gespeichert und danach der Durchschnitt berechnet und alle Werte ausgegeben. (Hinweis: Es wird davon ausgegangen, dass nur Noten von 1 bis 6 eingegeben werden.)

```
import java.util.*;
public class NotenAuswertung {
    public static void main(String[] args) {
        try {
            notenAuswerten();
        } catch (Exception ex) {
            System.out.println("Es ist ein Fehler aufgetreten " + ex);
        } finally {
            System.out.println("Das Programm wird beendet");
        }
    }

    public static void notenAuswerten() {
        Scanner input = new Scanner(System.in);
        System.out.println("Wieviele Noten wollen Sie eingeben?");
        int anzahl = input.nextInt();
        int[] noten = new int[anzahl];
        int[] anzahlProNote = new int[6];
        double summe = 0.0;
        for (int i = 0; i < anzahl; i++) {
            System.out.print("Geben Sie bitte die " + (i+1) +
                ".Note ein:");
            noten[i] = input.nextInt();
            switch(noten[i]) {
                case 1: anzahlProNote[0]++; break;
                case 2: anzahlProNote[1]++; break;
                case 3: anzahlProNote[2]++; break;
                case 4: anzahlProNote[3]++; break;
                case 5: anzahlProNote[4]++; break;
                case 6: anzahlProNote[5]++; break;
                default: break;
            }
            summe = summe + noten[i];
        }
        double durchschnitt = summe / anzahl;
        System.out.println("In der Klassenarbeit wurden " +
            anzahlProNote[0] + " Einsen geschrieben");
        System.out.println("In der Klassenarbeit wurden " +
            anzahlProNote[1] + " Zweien geschrieben");
        System.out.println("In der Klassenarbeit wurden " +
            anzahlProNote[2] + " Dreien geschrieben");
    }
}
```

TAG 14 – TAG 16 - Das objektorientierte Programmierparadigma in Java umsetzen

Die grundlegenden Konzepte des objektorientierten Programmierparadigmas sind Abstraktion, Kapselung, Vererbung und Polymorphie. Da Java eine rein objektorientierte Programmiersprache ist, werden diese Konzepte auch weitestgehend unterstützt. Allerdings wird die Mehrfachvererbung nur indirekt über Interfaces umgesetzt. Für die Beispiele in diesem Abschnitt wird von folgendem Szenario ausgegangen.

Ausgangssituation (vereinfachte Darstellung)

Im Rahmen eines Kundenauftrages für ein Luftfahrtunternehmen, welches verschiedene Arten von Luftfahrzeugen wie Flugzeuge oder auch Heißluftballons an Kunden vermietet, soll in der Entwicklungsabteilung von JIKU eine Verwaltungs- und Abrechnungssoftware erstellt werden. Im Vorfeld wurden schon verschiedene Klassen mithilfe von UML geplant, die nun in Java umgesetzt werden sollen. Zur Kontrolle der Funktionsweise der Klassen soll zunächst eine kleine Konsolenanwendung entwickelt werden.

LE 1 - Klassen, Eigenschaften, Methoden, Zugriffsmodifizier und Objekte

An einem einfachen Beispiel soll nun erläutert werden, wie Klassen und Objekte in Java programmiert werden, was Getter und Setter sind und welche Zugriffsmodifizier es gibt. Grundlage dafür bildet die in dem UML-Klassendiagramm dargestellte Klasse „Luftfahrzeug“.

Klasse

Eine Klasse beginnt in Java mit dem Schlüsselwort „*class*“. Danach wird der Klassenname angegeben. Die Eigenschaften bzw. die Methoden der Klasse werden dann innerhalb einer öffnenden und einer schließenden geschweiften Klammer definiert.

Luftfahrzeug	
- bezeichnung:	string
- gewicht:	double
- baujahr:	int
+ Luftfahrzeug()	
+ getBezeichnung():	string
+ setBezeichnung(string):	void
+ getGewicht():	double
+ setGewicht(double):	void
+ getBaujahr():	int
+ setBaujahr(int):	void
+ getDaten():	string

```
</> class Klassenname {    // Öffnende geschweifte Klammer
    // Eigenschaften
    // Methoden
}    // schließende geschweifte Klammer
```

Zugriffsmodifizier

Allen Komponenten (Eigenschaften und Methoden) einer Klasse sind Zugriffsmodifizier zugeordnet. Diese legen fest, welchen Klassen der Zugriff gestattet ist. Es werden hier nur die drei wesentlichen Zugriffsrechte betrachtet.

- **public:** Auf öffentliche (public) Komponenten darf uneingeschränkt zugegriffen werden.
- **private:** Ein Zugriff auf private Komponenten ist nur innerhalb der Klasse erlaubt.
- **protected:** Auf geschützte (protected) Komponenten kann nur innerhalb der eigenen Klasse und aus Klassen, die von dieser Klasse abgeleitet wurden, zugegriffen werden.

Dabei wird der Zugriffsmodifizier immer vor die jeweilige Eigenschaft oder Methode geschrieben. Wird kein Zugriffsmodifizier angegeben, ist der Zugriff auf die Komponente automatisch auf *private* gesetzt.

Zugriffsmodi- fier	Zugriff von fremden Klassen	Zugriff von abgeleite- ten Klassen	Zugriff innerhalb der Klasse
public	x	x	x
protected		x	x
private			x

```
</> private int anzahl;
protected string bezeichnung;
public double berechneKosten() { ... }
```

Beispiele für die Verwendung von Zugriffsmodifiern

Getter und Setter

Ein Gebot der objektorientierten Programmierung ist es, den direkten Zugriff auf die Eigenschaften eines Objekts von außen zu verhindern (Kapselung). Die Kapselung wird erreicht, indem man die Eigenschaften einer Klasse auf private setzt und über

```
</> private String bezeichnung;

public String getBezeichnung() {
    return this.bezeichnung;
}

public void setBezeichnung(String bezeichnung) {
    this.bezeichnung = bezeichnung;
}
```

get-Methoden (Getter) und set-Methoden (Setter) auf diese zugreift. Getter und Setter sind Methoden einer Klasse, mit deren Hilfe die Werte der Eigenschaften eines Objekts von außen gelesen bzw. gesetzt werden können. Sie stellen eine Art Schnittstelle dar. Der Vorteil besteht darin, dass bei notwendigen Änderungen nur der Quelltext der Schnittstellen angepasst werden muss, der Rest des Programms bleibt von den Änderungen unbeeinflusst. Im Beispiel ist die Implementierung des Getters und des Setters für die Eigenschaft „bezeichnung“ der Beispielsklasse „Luftfahrzeug“ aufgeführt.

Konstruktoren und Destruktoren

Ein Konstruktor ist die Methode einer Klasse, die automatisch beim Erzeugen eines Objektes aufgerufen wird. Konstruktoren werden meistens dafür benutzt,

```
</> public Luftfahrzeug() {
    this.bezeichnung = "unbekannt";
    this.gewicht = 0.0;
    this.baujahr = 0;
}
```

um das Objekt zu initialisieren, d. h. den Variablen Anfangswerte zuzuweisen oder weiteren Speicherplatz zu reservieren. In der Regel sollte ein Konstruktor public gesetzt werden, damit überhaupt ein Objekt erzeugt werden kann.

Das folgende Beispiel zeigt den Konstruktor der Klasse „Luftfahrzeug“ in Java implementiert. In Java muss ein Konstruktor den gleichen Namen wie die Klasse besitzen und hat keinen Rückgabotyp. Allerdings können einem Konstruktor verschiedene Übergabeparameter zugeordnet werden. Pro Klasse können mehrere Konstruktoren definiert werden, d.h., man kann sie in Java überladen.

Ein Destruktor ist jene Methode einer Klasse, die als Letztes vor dem Zerstören des Objektes aufgerufen wird. Destruktoren stellen das Gegenteil von Konstruktoren dar. Im Gegensatz zum Konstruktor kann es nur einen Destruktor pro Klasse geben. In der Regel werden Destruktoren dafür benutzt, um den vom Objekt belegten Speicher freizugeben. Da moderne Programmiersprachen wie Java einen sogenannten Garbage-Collector besitzen, der sich automatisch um die Freigabe von Speicher kümmert, werden Destruktoren kaum noch eingesetzt und müssen somit

auch nicht implementiert werden. Die Umsetzung der Klasse „Luftfahrzeug“ in Java würde wie folgt aussehen.

Umsetzung der Beispielklasse „Luftfahrzeug“

```
</> public class Luftfahrzeug {  
    //-----  
    // Attribute der Klasse mit entsprechenden Zugriffsmodifiern  
    //-----  
    private String bezeichnung;  
    private double gewicht;  
    private int baujahr;  
    //-----  
    // Konstruktors der Klasse  
    //-----  
    public Luftfahrzeug() {  
        this.bezeichnung = "unbekannt";  
        this.gewicht = 0.0;  
        this.baujahr = 0;  
    }  
    //-----  
    // Getter und Setter der Klasse  
    //-----  
    public String getBezeichnung() { return this.bezeichnung; }  
    public void setBezeichnung(String bezeichnung) { this.bezeichnung =  
                                                    bezeichnung; }  
    public double getGewicht() { return this.gewicht; }  
    public void setGewicht(double gewicht) { this.gewicht = gewicht; }  
    public int getBaujahr() { return this.baujahr; }  
    public void setBaujahr(int baujahr) { this.baujahr = baujahr; }  
    //-----  
    // Methoden  
    //-----  
    public String getData() {  
        String daten = "Bezeichnung: " + this.bezeichnung + "\n" +  
                      "Gewicht: " + this.gewicht + "kg \n" +  
                      "Baujahr: " + this.baujahr;  
        return daten;  
    }  
}
```

Objekte

Aus den Klassen können nun Objekte erzeugt werden. Hierfür wird das Schlüsselwort „new“ verwendet.

Allgemeine Syntax

```
</> <Klasse> <Objektbezeichnung> = new <Klasse>();
```

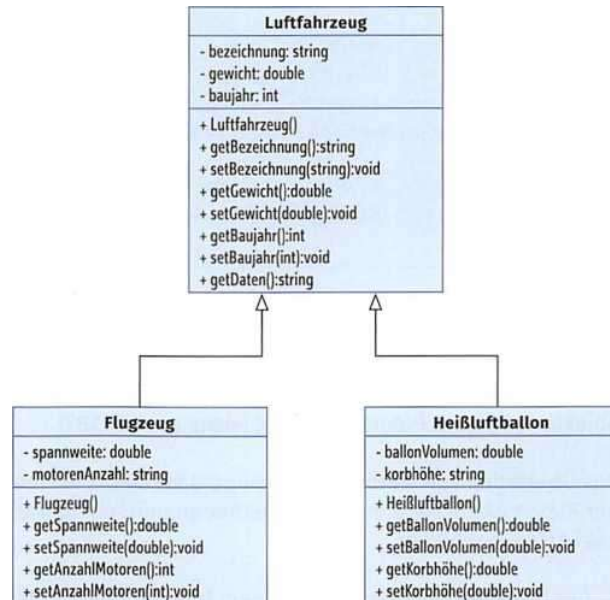
Zur Veranschaulichung dient das Beispiel „Erzeugen eines Objektes“ aus der Klasse „Luftfahrzeug“. Über die Getter und Setter des Objektes kann nun auf die Attribute zugegriffen werden. In dem Beispiel werden im Hauptprogramm dem Objekt konkrete Werte zugewiesen und danach die Daten mithilfe der Methode „getData()“ angezeigt.

Quelltext des Hauptprogramms „Luftfahrzeug“¹

```
</> public class Program {
    public static void main(String[] args) {
        Luftfahrzeug beispielLuftfahrzeug = new Luftfahrzeug();
        beispielLuftfahrzeug.setBezeichnung("A310");
        beispielLuftfahrzeug.setGewicht(71840);
        beispielLuftfahrzeug.setBaujahr(1995);
        System.out.print(beispielLuftfahrzeug.getDaten());
    }
}
```

LE 2 - Vererbung

Neben der Abstraktion und der Kapselung ist die Vererbung ein weiteres Basisprinzip der OOP. Über die Vererbung kann man eine neue Klasse auf Basis einer bereits vorhandenen Klasse erzeugen. Dabei wird zwischen Einfach- und Mehrfachvererbung unterschieden. Java erlaubt aus verschiedenen Gründen auf direktem Weg nur die Einfachvererbung. Eine Mehrfachvererbung kann nur indirekt mithilfe von Interfaces realisiert werden. Als Beispiel soll die Klasse „Luftfahrzeug“ um die Klassen „Flugzeug“ und „Heißluftballon“ erweitert werden. Diese Erweiterung ist im UML-Klassendiagramm dargestellt. Dabei erben die beiden neuen Klassen die Attribute und Methoden der Klasse „Luftfahrzeug“. Es kann aber nur auf die Methoden zugegriffen werden, da diese *public* sind. Die Eigenschaften sind als *private* gekennzeichnet und deswegen wird der Zugriff auf diese Methoden in den abgeleiteten Klassen verweigert. Soll der Zugriff möglich sein, müssen die Attribute das Zugriffsrecht „*protected*“ erhalten. Vererbung in Java wird durch das Schlüsselwort „*extends*“ realisiert. Die Umsetzung der Vererbung ist in dem nachstehenden Quellcode ersichtlich. Hier wird aber nur die Implementierung der beiden neuen Klassen aufgeführt.



Umsetzung der Beispielsklassen „Flugzeug“ und „Heißluftballon“

```
</> public class Flugzeug extends Luftfahrzeug {
    private double spannweite;
    private int motorenAnzahl;

    public Flugzeug() {
        this.spannweite = 0.0;
        this.motorenAnzahl = 0;
    }
}
```

```

    public double getSpannweite() { return this.spannweite; }
    public void setSpannweite(double spannweite) { this.spannweite =
                                                    spannweite; }

    public int getAnzahlMotoren() { return this.motorenAnzahl; }
    public void setAnzahlMotoren(int motorenAnzahl) { this.motorenAnzahl =
                                                    motorenAnzahl; }
}

public class Heißluftballon extends Luftfahrzeug {
    private double ballonVolumen;
    private double korbhöhe;

    public Heißluftballon() {
        this.ballonVolumen = 0.0;
        this.korbhöhe = 0.0;
    }

    public double getBallonVolumen() { return this.ballonVolumen; }
    public void setBallonVolumen(double ballonVolumen) { this.ballonVolumen =
                                                         ballonVolumen; }

    public double getKorbhöhe() { return this.korbhöhe; }
    public void setKorbhöhe(double korbhöhe) { this.korbhöhe = korbhöhe; }
}

```

LE 3 - Überladen und Überschreiben von Methoden

Das Überladen und Überschreiben von Methoden ist ein zentraler Bestandteil der Polymorphie in der objektorientierten Programmierung.

Das Überladen von Methoden bedeutet, dass mehrere Methoden mit demselben Methodennamen in einer Klasse existieren dürfen. Die Methoden müssen aber trotzdem zu unterscheiden sein. Dies kann auf zwei Arten geschehen:

1. Der Datentyp von mindestens einem Übergabeparameter ist anders als in den übrigen gleichnamigen Methoden.
2. Die Anzahl der Übergabeparameter ist unterschiedlich.

In dem folgenden Beispiel wurde der Konstruktor der Klasse „Flugzeug“ überladen. Es existieren drei Konstruktoren mit dem gleichen Namen, aber mit unterschiedlichen Übergabeparametern.

Überladen des Konstruktors in der Beispielklasse „Flugzeug“

```

</> public class Flugzeug extends Luftfahrzeug {
    ...
    public Flugzeug() { ... }
    public Flugzeug (String bezeichnung) { ... }
    public Flugzeug (String bezeichnung, double gewicht,
                    int baujahr, double spannweite, int motorenAnzahl) { ... }
    ...
}

```

Das Überschreiben von Methoden bedeutet, dass eine Methode einer Basisklasse in der abgeleiteten Klasse neu implementiert (überschrieben) wird. Die überschreibende Methode kann jedoch die überschriebene Methode aufrufen. Damit eine Methode überschrieben werden kann, müssen die folgenden Bedingungen erfüllt sein:

1. Die Methoden müssen namentlich übereinstimmen.
2. Die Methoden müssen im Datentyp und in der Anzahl ihrer Übergabeparameter übereinstimmen.
3. Der Rückgabetyp muss übereinstimmen.

4. Die überschreibende Methode darf in ihren Zugriffsrechten nicht stärker beschränkt sein als die überschriebene Methode. Die Zugriffsrechte dürfen jedoch „gelockert“ werden, z. B. von *protected* auf *public*.
5. Die zu überschreibende Methode muss auch tatsächlich vererbt werden. Auf *private* gesetzte Methoden werden nicht vererbt und können damit auch nicht überschrieben werden.

Das Überschreiben von Methoden wird an dem Beispiel der Methode „getDaten()“ aus der Klasse „Luftfahrzeug“ illustriert. Diese Methode wird von den Klassen „Flugzeug“ und „Heißluftballon“ geerbt. Allerdings gibt sie bisher nur die Werte für die Attribute „bezeichnung“, „gewicht“ und „baujahr“ aus. Damit alle Werte des entsprechenden Objektes ausgegeben werden, muss der Quelltext dieser Methode entsprechend erweitert werden. Dieses kann aber nicht in der Klasse „Luftfahrzeug“ geschehen, weil die abgeleiteten Klassen unterschiedliche Attribute besitzen. Also muss diese Methode in den neuen Klassen neu implementiert werden und damit wird gleichzeitig die geerbte Methode „getDaten()“ überschrieben.

Damit direkt wie im Beispiel dargestellt auf die Attribute der Klasse „Luftfahrzeug“ zugegriffen werden kann, müssen dort die Zugriffsmodifizier von *private* auf *protected* verändert werden. Alternativ könnten auch die entsprechenden Getter und Setter verwendet werden.

Überschreiben der Methode „getDaten()“ in den Beispielklassen „Flugzeug“ und „Heißluftballon“

```
</> public class Flugzeug extends Luftfahrzeug {
    ...
    public String getDaten() {
        String daten = "Bezeichnung: " + this.bezeichnung + "\n" +
            "Gewicht: " + this.gewicht + " kg \n" +
            "Baujahr: " + this.baujahr + "\n" +
            "Spannweite: " + this.spannweite + " kg \n" +
            "Anzahl Motoren: " + this.motorenAnzahl;

        return daten;
    }
    ...
}

public class Heißluftballon extends Luftfahrzeug {
    ...
    public String getDaten() {
        String daten = "Bezeichnung: " + this.bezeichnung + "\n" +
            "Gewicht: " + this.gewicht + " kg \n" +
            "Baujahr: " + this.baujahr + "\n" +
            "Volumen des Ballons: " + this.ballonVolumen + " m3\n" +
            "Korbhöhe: " + this.korbhöhe + " m";

        return daten;
    }
    ...
}
```

LE 4 - Abstrakte Klasse und Interface

Abstrakte Klasse

Eine abstrakte Klasse ist eine spezielle Klasse, die neben normalen Methoden auch sogenannte abstrakte Methoden enthält. Abstrakte Methoden sind Methoden, die eine Methodensignatur, aber keinen Methodenrumpf besitzen, abstrakte Methoden sind nicht implementiert und können deshalb auch nicht ausgeführt werden. Aus diesem Grund ist es auch nicht möglich, von abstrakten Klassen Objekte zu bilden. Oder anders ausgedrückt: Es ist nicht möglich, abstrakte Klassen zu instanziierten. Der Sinn und Zweck einer abstrakten Klasse bestehen darin,

dass von ihr geerbt wird. Die nicht implementierten Methoden müssen dann in der neuen Klasse „ausprogrammiert“, also überschrieben werden. Erst dann ist es möglich, eine Instanz von der neuen Klasse zu bilden. Verwendet werden abstrakte Klassen beispielsweise als Basis-Klassen, um grundlegende Eigenschaften und Methoden ihrer Unterklassen festzulegen, ohne diese bereits konkret zu implementieren. Zur Veranschaulichung soll das folgende Beispiel dienen.

Abstrakte Klasse „Luftfahrzeug“

```
</> abstract class Luftfahrzeug {  
    ...  
    public abstract String getDaten(); // abstrakte Methode  
    ...  
}
```

In dem Beispiel wird aus der normalen Klasse „Luftfahrzeug“ eine abstrakte Klasse, indem zunächst einmal der Klassendefinition das Schlüsselwort „abstract“ vorangestellt wird. Weiterhin wird die Methode „getDaten()“ in eine abstrakte Methode umgewandelt, in dem hier ebenfalls das Schlüsselwort „abstract“ verwendet und kein Methodenrumpf implementiert wird. Der Rest des Programms bleibt gleich. Da nun von der Klasse „Luftfahrzeug“ kein Objekt mehr angelegt werden kann, muss das Hauptprogramm entsprechend angepasst werden. Dazu werden jeweils ein Flugzeug und ein Heißluftballon erzeugt und mit Daten versehen, welche im Anschluss wieder ausgegeben werden.

Angepasstes Hauptprogramm

```
</> public class Program {  
    public static void main(String[] args) {  
        System.out.println("Beispielflugzeug");  
        Flugzeug f = new Flugzeug();  
        f.setBezeichnung("A310");  
        f.setGewicht(71840);  
        f.setBaujahr(1995);  
        f.setSpannweite(43.90);  
        f.setAnzahlMotoren(2);  
        System.out.print(f.getData());  
        System.out.println("\n");  
        System.out.println("Beispielheißluftballon");  
        Heißluftballon b = new Heißluftballon();  
        b.setBezeichnung("Ultra Magic");  
        b.setGewicht(478.5);  
        b.setBaujahr(2018);  
        b.setBallonVolumen(4000);  
        b.setKorbhöhe(1.10);  
        System.out.print(b.getData());  
    }  
}
```

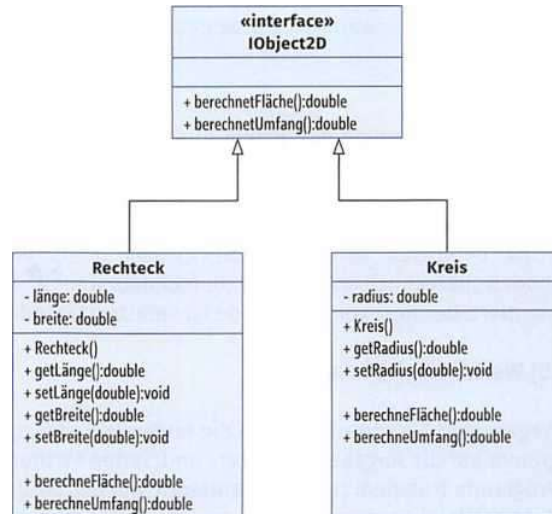
Interface

Ein Interface (Schnittstelle) in Java enthält ausschließlich abstrakte Methoden und Konstanten. Wird eine Klasse aus einem Interface abgeleitet, müssen alle Methoden des Interfaces in der neuen Klasse implementiert werden. Es ist auch möglich, dass eine Klasse nicht alle Methoden des Interfaces implementiert. In diesem Fall muss die Klasse aber als abstrakt gekennzeichnet werden. In den folgenden Punkten unterscheidet sich ein Interface von einer abstrakten Klasse:

- In einem Interface sind alle Methoden abstrakt, während eine abstrakte Klasse auch implementierte Methoden enthalten kann.
- In einem Interface ist die Sichtbarkeitsstufe für alle Elemente immer *public*, in abstrakten Klassen können auch andere Sichtbarkeitsstufen verwendet werden.

- Abstrakte Klassen werden als Klassen gesehen und somit kann in Java nur von einer abstrakten Klasse gleichzeitig geerbt werden. Im Gegensatz dazu ist es möglich, dass eine Klasse von mehreren Interfaces erbt und diese implementiert.

Da das Beispiel mit den Luftfahrzeugen zu stark umgebaut werden müsste, wird an dieser Stelle ein anderes Beispiel verwendet. In diesem wird ein Interface „IObject2D“ angelegt, das die abstrakten Methoden „berechneFläche“ und „berechneUmfang“ definiert. Diese Methoden müssen in allen abgeleiteten Klassen implementiert werden, im konkreten Fall in den Klassen „Rechteck“ und „Kreis“.



Implementierung eines Interface in Java

```

</> public interface IObjekt2D {
    public double berechneFläche();
    public double berechneUmfang();
}

public class Rechteck implements IObjekt2D {
    private double laenge;
    private double breite;

    public double getLaenge() { return this.laenge; }
    public void setLaenge( double laenge ) { this.laenge = laenge; }
    public double getBreite() { return this.breite; }
    public void setBreite( double breite ) { this.breite = breite; }

    public double berechneFläche() { return (laenge * breite); }
    public double berechneUmfang() { return ((laenge + breite) * 2); }
}

public class Kreis implements IObjekt2D {
    private double radius;

    public double getRadius() { return this.radius; }
    public void setRadius( double radius ) { this.radius = radius; }

    public double berechneFläche() { return (Math.PI * radius * radius); }
    public double berechneUmfang() { return (2 * Math.PI * radius); }
}
    
```

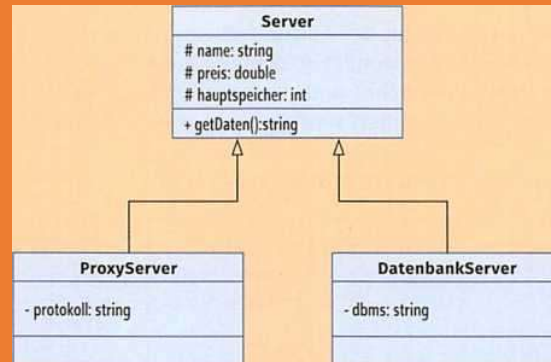
Die Implementierung von einem Interface ist in Java relativ einfach. Zunächst wird das Schlüsselwort „class“ durch das Schlüsselwort „interface“ ersetzt. Es ist zum Standard geworden, dass vor den Interface-Namen ein großgeschriebenes „I“ gesetzt wird, z. B. „I MeinInterface“. Wenn von einem Interface eine neue Klasse abgeleitet wird, geschieht dies in Java durch das Schlüsselwort „implements“ anstatt dem bisher benutzten „extends“. In den Klassen „Kreis“ und „Rechteck“ werden dann jeweils die beiden Methoden „berechneFläche()“ und „berechneUmfang()“ entsprechend den mathematischen Formeln implementiert. Der Rest vom Quellcode ist selbsterklärend.

LE 5 - Weitere Beispielaufgabe

Wegen ihrer Komplexität wurde die folgende Beispielaufgabe reduziert. Weiterhin wurde im Klassendiagramm auf die Angabe der Getter- und Setter-Methoden und des Konstruktors verzichtet. Diese sind im Programm trotzdem zu implementieren. Bei der Lösung wird auch nur die Implementierung der Klassen dargestellt. Ein entsprechendes Hauptprogramm muss noch entwickelt werden.

Beispielaufgabe

In der Planung für ein Java-Programm zur Serververwaltung wurde ein erstes Klassendiagramm entwickelt. Implementieren Sie dieses Klassendiagramm in Java. Geben Sie dabei alle Getter- und Setter-Methoden an. Außerdem soll jede Klasse einen Konstruktor erhalten, in dem die Eigenschaften initialisiert werden.



Lösung

Zunächst wird die Klasse „Server“ implementiert. Die Eigenschaften dieser Klasse werden alle als *protected* implementiert, damit sie später vererbt werden können.

```
public class Server {
    protected String name;
    protected double preis;
    protected int hauptspeicher;

    public Server() {
        this.name = "unbekannt";
        this.preis = 0.0;
        this.hauptspeicher = 0;
    }

    public String getName() { return this.name; }
    public void setName(String name) { this.name = name; }
    public double getPreis() { return this.preis; }
    public void setPreis(double preis) { this.preis = preis; }
    public int getHauptspeicher() { return this.hauptspeicher; }
```

```
public void setHauptspeicher(int hauptspeicher) { this.hauptspeicher =
                                                    hauptspeicher; }

//-----
// Methoden
//-----
public String getData() {
    String daten = "Name: " + this.name + "\n" +
                  "Preis: " + this.preis + "\n" +
                  "Hauptspeicher: " + this.hauptspeicher;

    return daten;
}
}
```

Aus dieser Klasse werden nun die beiden anderen Klassen abgeleitet.

```
public class ProxyServer extends Server {
    private String protokoll;

    public ProxyServer() {
        this.protokoll = "unbekannt";
    }
    public String getProtokoll() { return this.protokoll; }
    public void setProtokoll(String protokoll) { this.protokoll =
                                                protokoll; }
}

public class DatenbankServer extends Server {
    private String dbms;

    public DatenbankServer() {
        this.dbms = "unbekannt";
    }
    public String getDBMS() { return this.dbms; }
    public void setDBMS(String dbms) { this.dbms = dbms; }
}
```

TAG 17 & TAG 18 - Grafische Benutzerschnittstellen in Java entwickeln

Die Programmierung einer grafischen Benutzeroberfläche in Java ist komplex. Der wesentliche Unterschied zwischen grafischer Benutzeroberfläche und Konsolenanwendung besteht in der unterschiedlichen Interaktion des Benutzers mit dem Programm, d.h., es sind z. B. die Ein- und Ausgabe verschieden. Eine grafische Benutzeroberfläche bietet hier viel mehr Möglichkeiten der Interaktion, nicht zuletzt durch eine sehr große Anzahl an Oberflächenelementen. Das Entwerfen und Gestalten von Oberflächen, die erwartungskonform und benutzerfreundlich sind, ist vielfach mit erheblichem Aufwand verbunden. Der folgende Abschnitt beschränkt sich darauf, die Grundlagen anhand von einfachen Beispielen zu erläutern.



LE 1 - Grundlagen

Eine Programmiersprache wie Java, die zum Ziel hat, plattformunabhängige Softwareentwicklung zu unterstützen, muss auch eine Bibliothek anbieten, mit der sich grafische Oberflächen gestalten lassen. Generell müssen solche grafischen Bibliotheken folgende drei grundlegende Elemente abdecken.

- Sie muss grafische Interaktionskomponenten, die sogenannten Widgets, wie Fenster, Schaltflächen, Textfelder, Menüs und Container unterstützen.
- Sie muss ein Modell zur Behandlung von Ereignissen definieren.
- Sie muss grafische Primitivoperationen wie Linien und Polygone darstellen und Farben und Zeichensätze zuweisen können.

Die erste API, welche von Java in dieser Richtung zur Verfügung gestellt wurde, war das Abstract Window Toolkit (AWT). Nach der Freigabe des ersten AWT wurde jedoch schnell klar, dass das Leistungsangebot keineswegs für die Zukunft reicht. Deswegen wurden Erweiterungen geschaffen, die unter dem Namen Java Foundation Classes (JFC) in Java eingeführt wurden. Sie enthalten u.a. die Swing GUI-Komponenten, welche bis 2014 in Java der Standard waren und schließlich durch die neu entwickelte moderne API JavaFX abgelöst werden sollten. Dieses geschah aber nicht in dem geplanten Umfang. Java Swing wird auch heutzutage noch sehr stark genutzt und existiert parallel zu JavaFX. Deswegen wird an dieser Stelle zunächst nur Java Swing näher betrachtet. Auf JavaFX wird dann im dritten Jahrgangsband für die Fachinformatiker Anwendungsentwicklung näher eingegangen.

Grundlegende Bestandteile der JFC

„Swing“ GUI-Komponenten

Swing-Komponenten sind im Gegensatz zu den AWT-Komponenten neue grafische Elemente, welche komplett in Java implementiert wurden und plattformunabhängig sind. Der Name „Swing“ war ein Projektname dieser Komponenten. Obwohl sie nun Teil der JFC sind, blieb der Name bestehen.

Pluggable Look and Feel

Darunter ist zu verstehen, dass das Aussehen der Komponenten zur Laufzeit geändert werden kann, ohne dass das Programm neu gestartet werden muss. Alle Komponenten des Swing-Sets besitzen diese Fähigkeit automatisch.

Accessibility

Hiermit ist die Unterstützung der Barrierefreiheit gemeint. Mit dieser API kann mit vielfältigen Interaktionstechniken auf die JFC- und AWT-Komponenten zugegriffen werden. Zu diesen Techniken zählen unter anderem Lesegeräte für Blinde, eine Lupe für den Bildschirm oder auch die Spracherkennung.

Java 2D API

Die 2D-Funktionsbibliothek ist eine Technik, die über eine Objektbeschreibung Objekte bildet und diese auf dem Bildschirm darstellt.

Der grundlegendste Bestandteil einer grafischen Benutzeroberfläche ist das Fenster (Frame, Window). Daher wird im Folgenden zunächst darauf eingegangen, wie solch ein Fenster unter Java erzeugt und angezeigt wird. Dazu wird in dem Beispiel die Klasse „JFrame“ verwendet, welche durch die entsprechende Bibliothek in das Programm mit eingebunden wird. Der Quellcode für das Programm mit den entsprechenden Erklärungen ist nachfolgend aufgeführt. Alle nicht kommentierten Sprachelemente aus dem Quelltext sollten aus den vorhergehenden Abschnitten über Java bekannt sein.

Beispiel für ein Basisfenster in Java

```
</> import javax.swing.JFrame;    // die Klasse JFrame ins Programm einbinden

public class Program {
    public static void main(String[] args) {
        try {
            JFrame mainFrame = new JFrame()           // ein neues Fenster erstellen
            mainFrame.setTitle("Das erste Java-Fenster ..."); // Titel
            mainFrame.setSize(800, 400);               // Größe des Fensters festlegen
            mainFrame.setLocation(400, 200);           // Position auf dem Bildschirm
            mainFrame.setVisible(true);                // Fenster anzeigen
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Nach dem Starten des Programms ist das abgebildete Fenster zu sehen.

Das Fenster selbst ist unspektakulär, es besitzt aber bereits die Grundfunktionalität eines jeden Fensters. Es kann beispielsweise vergrößert und verkleinert, verschoben oder geschlossen werden. Als nächster Schritt müssen nun die einzelnen Oberflächenelemente und deren Funktionalität dem Fenster hinzugefügt werden. Dieses kann so erfolgen, dass der Programmierer dafür den entsprechenden Quelltext schreibt. Allerdings ist diese Verfahrensweise für ergonomisch zu gestaltende Oberflächen mit vielen Widgets sehr zeitaufwendig. Zum Beispiel müssen alle Koordinaten für die Größe und die Position für alle Komponenten ermittelt und eingegeben werden. Auch ein Ändern und Ausprobieren von Alternativen ist nicht auf die Schnelle möglich.



Programmausschnitt für einen Button, welcher auf einen Klick das Programm schließt

```
</> import javax.swing.JButton;
import java.awt.event.ActionListener;

...
JButton btnBeenden = new JButton("Beenden");
btnBeenden.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});
btnBeenden.setBounds(300, 200, 100, 25);
...
```

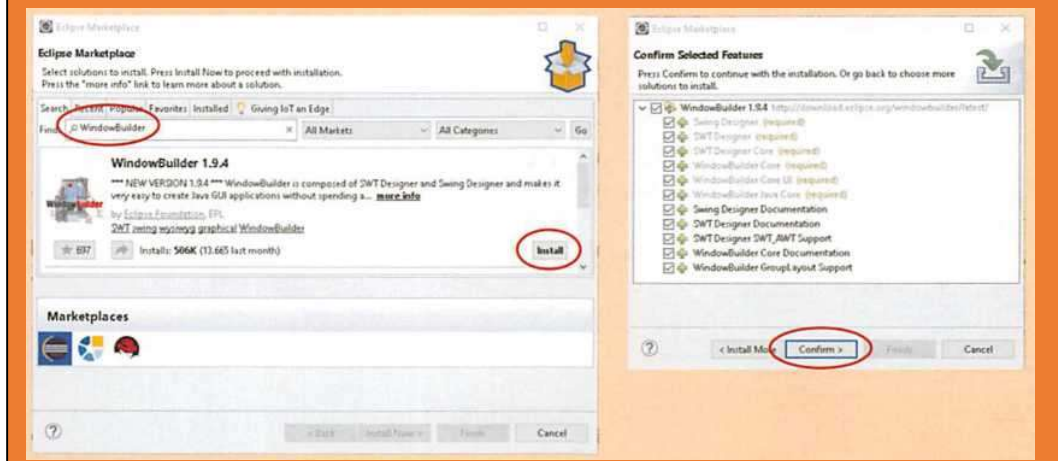
Eine Alternative zu dieser Form von händischem Programmieren von Benutzerschnittstellen stellen GUI-Builder dar. Ein GUI-Builder (GUI-Designer, GUI-Editor) ist ein Programmierwerkzeug, mit dessen Hilfe die Entwicklung von grafischen Benutzerschnittstellen vereinfacht wird. Der GUI-Builder ermöglicht es, dass Steuerelemente per Drag and Drop auf einer grafischen Oberfläche angeordnet werden. Außerdem können die Eigenschaften der Steuerelemente leicht geändert und Ereignisse, z.B. Klick oder Doppelklick, definiert werden. Der dazugehörige Quelltext wird automatisch erzeugt und angepasst. Somit kann der Programmierer viel Zeit sparen, in der er sich auf wesentlichere Programminhalte konzentrieren kann. Die GUI-Builder sind entweder schon standardmäßig in der Entwicklungsumgebung (IDE) integriert oder sie können in den meisten Fällen nachträglich installiert werden. Die letztere Variante gilt auch für Eclipse. Der GUI-Builder hat in diesem Fall den Namen „WindowBuilder“ und kann über den Eclipse Marketplace direkt in Eclipse eingebunden werden.

LE 2 - Nutzung des WindowBuilder

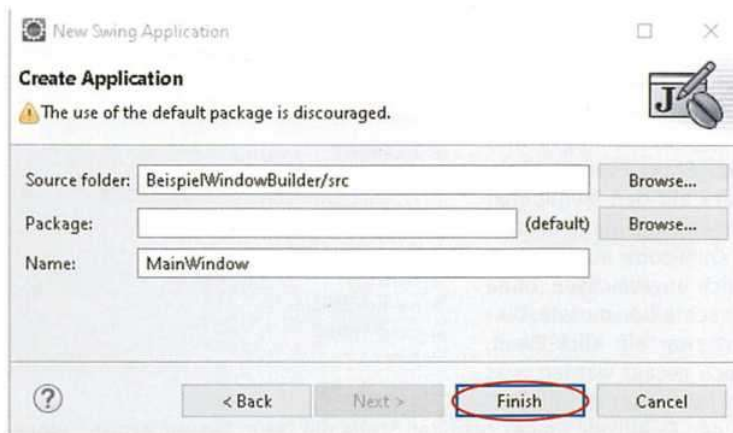
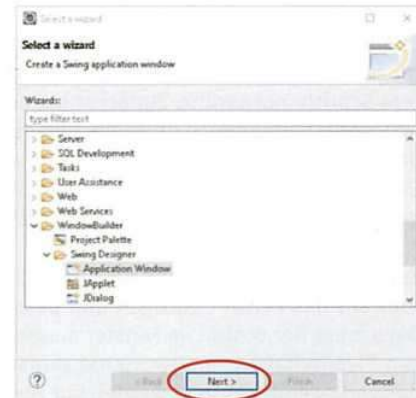
Für die nächsten Beispiele wird nur noch der WindowBuilder zur Oberflächengestaltung verwendet. Deswegen folgt eine kurze Einführung in den Umgang mit dem WindowBuilder.

Einbinden vom WindowBuilder in Eclipse

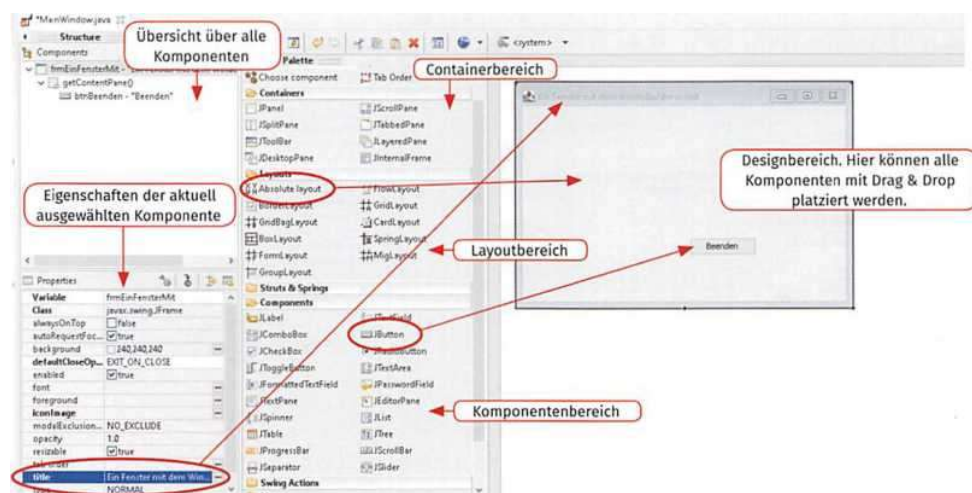
1. Aufrufen des Menüpunktes Help - Eclipse Marketplace.
2. In dem Dialog unter dem Menüpunkt „Search“ - Find: den Begriff „WindowBuilder“ eingeben und danach suchen lassen. Danach mithilfe des Install-Buttons den WindowBuilder installieren.
3. Im nächsten Dialog die benötigten Features auswählen und mit dem Confirm-Button die Auswahl bestätigen.
4. Im anschließenden Dialog „Restart now“ einen Neustart der Entwicklungsumgebung veranlassen; danach steht der WindowBuilder in Eclipse zur Verfügung.



Ist der WindowBuilder eingebunden, kann damit ein entsprechendes Java-Projekt entwickelt werden. Als Beispiel soll wieder ein Windowsfenster mit einem einfachen Beenden-Button dienen, welches aber dieses Mal ausschließlich mit dem WindowBuilder erstellt wird. Zunächst wird wieder ein neues Projekt angelegt und danach dem Projekt aus dem WindowBuilder ein „Application Window“ hinzugefügt. Dazu muss nebenstehender Katalog über die Menüpunkte File - New - Others (ganz unten) aufgerufen werden. Nach Betätigen des Buttons „Next“ erscheint ein weiteres Fenster „Create Application“, in dem der Name des Application Window eingegeben werden kann. In diesem Fall wird der Name „MainWindow“ verwendet.



Nach dem Bestätigen mit dem Finish-Button wird der Quellcode für das Fenster automatisch erstellt und angezeigt. Dieses ist zwar schon ein gewaltiger Fortschritt, aber um mit Drag and Drop Steuerelemente zu platzieren, muss noch in die Design-Ansicht gewechselt werden. Dazu muss unter dem Quellcode von „Source“ auf „Design“ geklickt werden und es erscheint danach folgende Ansicht.



Es gibt dort verschiedene Bereiche (siehe Beschreibung in Abbildung). Zu Beginn ist nur ein leeres Fenster zu sehen und nicht bereits ein Fenster mit Button, wie es auf dem Bild dargestellt ist. Dazu sind weitere Schritte notwendig. Zunächst bekommt das Fenster einen entsprechenden Fenstertitel. Dieser wird im Eigenschaftsbereich unter der Eigenschaft „title“ eingegeben. Danach wird aus dem Bereich „Layouts“ das „Absolute Layout“ ausgewählt und einfach auf das

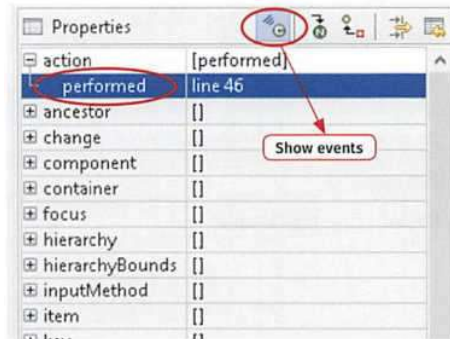
Fenster gezogen. Mit diesem Layout könnten alle Steuerelemente frei auf dem Fenster platziert werden. Die anderen Layouts haben auch ihre Berechtigung, aber können hier nicht ausführlicher behandelt werden.

Zum Schluss wird noch aus dem Bereich „Components“ ein „JButton“ auf das Fenster gezogen und dessen Eigenschaften gesetzt. Dazu muss der Button im Fenster ausgewählt werden, sodass dessen Eigenschaften im Eigenschaftsfenster angezeigt werden. Für das vorliegende Beispiel werden nur zwei Eigenschaften geändert.

Eigenschaft	Wert
Variable	btnClose
text	Beenden

Bei der einen Eigenschaft handelt es sich um den Objektnamen des Buttons und bei der anderen um den Text, welcher auf dem Button angezeigt wird.

Für alle anderen Eigenschaften bleiben erst einmal die Ausgangswerte bestehen und werden später bei Bedarf entsprechend angepasst. Zu guter Letzt muss noch ein Klick-Ereignis für den Button erzeugt werden. Dazu wird im Eigenschaftsfenster der Punkt „Show Events“ ausgewählt und durch einen Doppelklick auf den Menüpunkt „action“ ein entsprechendes Event erzeugt. Bis zu diesem Zeitpunkt wurde der komplette Quellcode automatisch angelegt und ist schon beträchtlich angewachsen, ohne dass man eine einzige Zeile selbst schreiben musste. Dieses ändert sich jetzt. Es existiert zwar ein Klick-Event, aber es muss dem Programm noch gesagt werden, was daraufhin geschehen soll. Dazu wird wieder auf die Ansicht „Source“ gewechselt und in den Quellcode an der richtigen Stelle die Zeile „System.exit(0);“ eingefügt. Dieses ist im folgenden Quellcodeausschnitt nochmals verdeutlicht.



ActionListener für den Beenden-Button

```

</>
...
JButton btnClose = new JButton("Beenden");
btnBeenden.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});
...

```

Diese Zeile wird eingefügt

Hierbei ist zu beachten, dass bis auf die eine Zeile alles andere an Quelltext schon vorhanden ist. Danach kann das Programm gestartet werden. Das Ergebnis ist bereits bekannt. Der Unterschied besteht nur darin, dass die Erstellung der Oberfläche mit dem Window-Builder deutlich einfacher ist.



Weitere Beispielaufgabe

Das Beispiel ist bereits aus dem Abschnitt der Konsolenprogrammierung mit Java bekannt. Es sollen zwei Zahlen addiert und das Ergebnis ausgegeben werden. Anhand dieses Beispiels wird gezeigt, wie das Programm mit einer grafischen Benutzerschnittstelle realisiert werden kann.

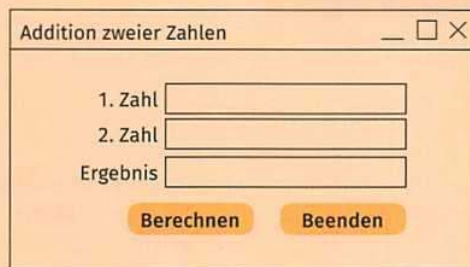
Beispielaufgabe

Es soll ein Programm mit einer grafischen Benutzeroberfläche entwickelt werden, welches nach der Eingabe zweier Zahlen diese addiert und das Ergebnis ausgibt.

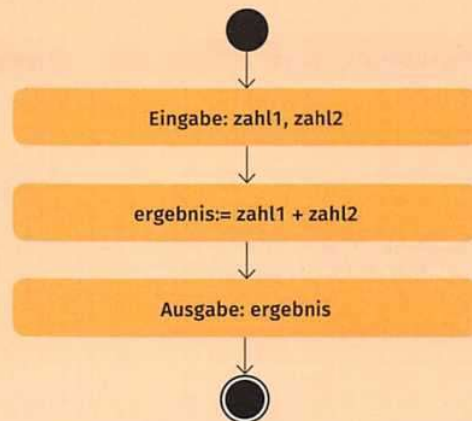
Lösung

Zunächst werden die Oberfläche und der Algorithmus geplant. Für diese Aufgabe könnte man zwar diese Phase überspringen und gleich zur Umsetzung in Java übergehen. Allerdings soll hier an einem einfachen Beispiel der vollständige Entwicklungsprozess aufgezeigt werden, ohne ins Detail zu gehen.

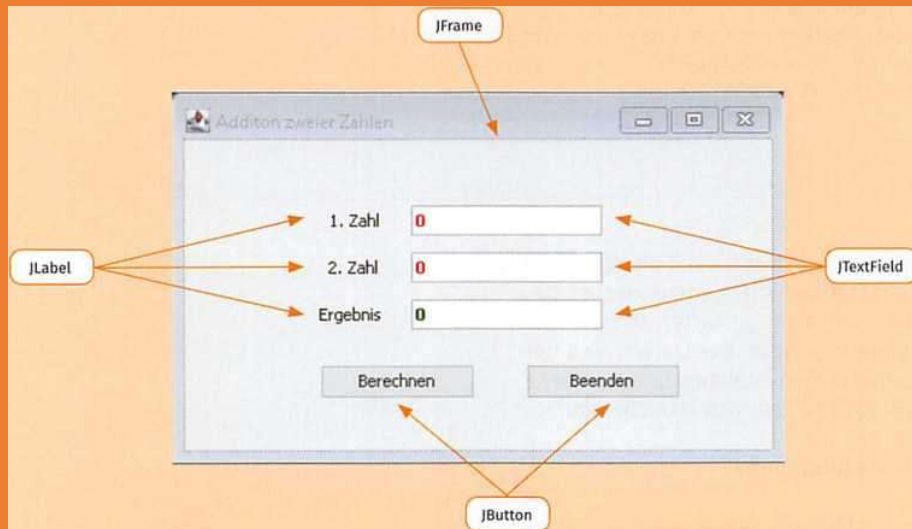
Mockup der Oberfläche



Algorithmus als Aktivitätsdiagramm



Nach der Planung erfolgt nun die Umsetzung in Java mithilfe von Eclipse. Dazu wird im WindowBuilder ein entsprechendes Fenster erzeugt und die Widgets platziert und deren Eigenschaften gesetzt. Eine mögliche Lösung wird hier vorgestellt.



Eigenschaften, welche bei den einzelnen Steuerelementen geändert wurden, sind folgende: Nachdem die Steuerelemente im WindowBuilder gesetzt und initialisiert wurden, werden nun die Klick-Events für die beiden Buttons erzeugt und der notwendige Quellcode implementiert. Nachfolgend ist der gesamte Quellcode der Anwendung dargestellt und die Stellen, welche eingegeben werden müssen, sind extra gekennzeichnet.

Frame		Labels	
Eigenschaft	Wert	Eigenschaft	Wert
Variable	frmMain	text	1. Zahl
title	Addition zweier Zahlen	text	Ergebnis
		Eigenschaft	Wert
		text	2. Zahl
Buttons		TextFields	
Eigenschaft	Wert	Eigenschaft	Wert
Variable	btnClose	Variable	txtValue1
text	Beenden	text	0
		foreground	220,20,60
Eigenschaft	Wert	Eigenschaft	Wert
Variable	btnClose	Variable	txtValue2
text	Beenden	text	0
		foreground	220,20,60
Eigenschaft	Wert	Eigenschaft	Wert
Variable	txtResult	text	0
text	0	foreground	0,128,0

Quellcode des Programms

```

import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JTextField;
import javax.swing.JLabel;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Color;
import java.awt.Font;

public class MainWindow {
    private JFrame frmMain;
    private JTextField txtValue1;
    private JTextField txtValue2;
    private JTextField txtResult;

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    MainWindow window = new MainWindow();
                    window.frmMain.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    public MainWindow() {
        initialize();
    }

    private void initialize() {
        frmMain = new JFrame();
        frmMain.setTitle("Addition zweier Zahlen");
        frmMain.setBounds(100, 100, 409, 246);
        frmMain.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frmMain.getContentPane().setLayout(null);
        JButton btnCalc = new JButton("Berechnen");
        btnCalc.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                double zahl1, zahl2, ergebnis;
                zahl1 = Double.parseDouble(txtValue1.getText());
                zahl2 = Double.parseDouble(txtValue2.getText());
                ergebnis = zahl1 + zahl2;
                txtResult.setText(String.valueOf(ergebnis));
            }
        });
        btnCalc.setBounds(91, 149, 103, 23);
        frmMain.getContentPane().add(btnCalc);

        JButton btnClose = new JButton("Beenden");
        btnClose.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
        btnClose.setBounds(228, 149, 103, 23);
        frmMain.getContentPane().add(btnClose);
    }
}

```

Notwendige Bibliotheken importieren

Main-Methode
Anlegen und Anzeigen
des Hauptfensters

Konstruktor

Fenster anlegen
und initialisieren

Button
„Berechnen“
anlegen und
initialisieren

Dieser Teil muss
implementiert
werden.

Umsetzung des
Algorithmus in Java

Button
„Beenden“
anlegen und
initialisieren

Dieser Teil muss
implementiert werden.

```

txtValue1 = new JTextField();
txtValue1.setFont(new Font("Tahoma", Font.BOLD, 11));
txtValue1.setForeground(new Color(220, 20, 60));
txtValue1.setText("0");
txtValue1.setBounds(152, 44, 127, 20);
frmMain.getContentPane().add(txtValue1);
txtValue1.setColumns(10);

txtValue2 = new JTextField();
txtValue2.setFont(new Font("Tahoma", Font.BOLD, 11));
txtValue2.setForeground(new Color(220, 20, 60));
txtValue2.setText("0");
txtValue2.setColumns(10);
txtValue2.setBounds(152, 75, 127, 20);
frmMain.getContentPane().add(txtValue2);

txtResult = new JTextField();
txtResult.setFont(new Font("Tahoma", Font.BOLD, 11));
txtResult.setText("0");
txtResult.setForeground(new Color(0, 128, 0));
txtResult.setColumns(10);
txtResult.setBounds(152, 106, 127, 20);
frmMain.getContentPane().add(txtResult);

JLabel lblNewLabel = new JLabel("1. Zahl");
lblNewLabel.setBounds(98, 47, 55, 14);
frmMain.getContentPane().add(lblNewLabel);

JLabel lblZahl = new JLabel("2. Zahl");
lblZahl.setBounds(97, 77, 51, 17);
frmMain.getContentPane().add(lblZahl);

JLabel lblErgebnis = new JLabel("Ergebnis");
lblErgebnis.setBounds(91, 109, 62, 14);
frmMain.getContentPane().add(lblErgebnis);
    }
}

```

Anlegen und
Initialisieren
der Textfelder

Anlegen und
Initialisieren
der Labels

Nachdem alles korrekt implementiert wurde, kann das Programm gestartet und getestet werden.

Beispielausgabe:



TAG 19 & TAG 20 - Anwendungen in Python implementieren

In diesem Kapitel sollen die Kenntnisse in Python bezüglich der objektorientierten Programmierung und der Umsetzung von grafischen Benutzeroberflächen erweitert werden. Für die Inhalte dieses Kapitels sind grundlegende Kenntnisse in Python und einer entsprechenden Entwicklungsumgebung, z.B. PyCharm, Voraussetzung (zu den Grundlagen von Python).



LE 1 - Das objektorientierte Programmierparadigma in Python umsetzen

Die grundlegenden Konzepte des objektorientierten Programmierparadigmas sind Abstraktion, Kapselung, Vererbung und Polymorphie. Da Python keine rein objektorientierte Programmiersprache ist, werden manche Konzepte, z. B. das Überladen von Methoden, nur teilweise unterstützt.

Ausgangssituation (vereinfachte Darstellung)

Im Rahmen eines Kundenauftrags von einem Musik-Store soll die Entwicklungsabteilung von JIKU eine Software erstellen, mit der u.a. verschiedene Musikinstrumente verwaltet werden. Im Vorfeld wurden schon verschiedene Klassen mithilfe von UML geplant, die nun in Python umgesetzt werden sollen. Zur Kontrolle der Funktionsweise der Klassen soll zunächst eine kleine Konsolenanwendung entwickelt werden.

LE 2 - Klassen, Eigenschaften, Methoden, Zugriffsmodifizier und Objekte

An einem einfachen Beispiel soll nun erläutert werden, wie Klassen und Objekte in Python programmiert werden, was Getter und Setter sind und welche Zugriffsmodifizier es gibt. Grundlage dafür bildet die in dem UML-Klassendiagramm dargestellte Klasse „Musikinstrument“.

Klasse

Eine Klasse in Python beginnt mit dem Schlüsselwort `class`. Danach wird der Klassenname angegeben und die Definition mit einem Doppelpunkt abgeschlossen.

Zugriffsmodifizier

Allen Komponenten (Eigenschaften und Methoden) einer Klasse sind Zugriffsmodifizier zugeordnet. Diese legen fest, welchen der Klassen und abgeleiteten Klassen der Zugriff gestattet ist. Python unterstützt drei Sichtbarkeitsstufen.

Musikinstrument
- modell: string - hersteller: string - preis: double
+ Musikinstrument() + setModell(string):void + getModell():string + setHersteller(string):void + getHersteller():string + setPreis(double):void + getPreis():double + getDate():string

```
</> class Klassenname:
    # Eigenschaften
    # Methoden
```

Zugriffsmodifizier	Beschreibung
--------------------	--------------

public	Auf öffentliche (<i>public</i>) Komponenten darf uneingeschränkt zugegriffen werden. Ohne spezielle Kennzeichnung sind alle Attribute und Methoden in Python <i>public</i> . Dies ist in Python die Default-Einstellung.
private	Ein Zugriff auf private-Komponenten ist nur innerhalb der Klasse erlaubt. Attribute und Methoden werden als <i>private</i> deklariert, indem man ihren Namen mit dem Präfix <code>____</code> (doppelter Unterstrich) versieht.
protected	Attribute und Methoden, welche mit nur einem Unterstrich (<code>_</code>) beginnen, werden als <i>protected</i> betrachtet. Technisch gesehen verhalten sich diese aber wie als <i>public</i> deklarierte Attribute und Methoden. Es soll lediglich darauf hinweisen, keinen direkten Zugriff von außen zu implementieren.

Beispiele für die Verwendung von Zugriffsmodifiern

```
</>
anzahl = 0      # public
__anzahl = 0    # private
_anzahl = 0     # protected

def berechne_kosten(): # public
    pass
def __berechne_kosten(): # private
    pass
def _berechne_kosten(): # protected
    pass
```

Getter und Setter

Ein Gebot der objektorientierten Programmierung ist es, den direkten Zugriff auf die Eigenschaften eines Objekts von außen zu verhindern (Kapselung). Die Kapselung wird erreicht, indem man die Eigenschaften einer Klasse auf *private* setzt und über get-Methoden (Getter) und set-Methoden (Setter) auf diese zugreift. Getter und Setter sind Methoden einer Klasse, mit deren Hilfe die Werte der Eigenschaften eines Objekts von außen gelesen bzw. gesetzt werden können. Sie stellen eine Art Schnittstelle dar. Der Vorteil besteht darin, dass bei notwendigen Änderungen nur der Quelltext der Schnittstellen angepasst werden muss, der Rest des Programms bleibt von den Änderungen unbeeinflusst. Im Beispiel ist die Implementierung des Getters und des Setters für die Eigenschaft „modell“ der Beispielklasse „Musikinstrument“ aufgeführt.

```
</>
class Musikinstrument:
    __modell = "unbekannt"
    def get_modell(self): # Getter
        return self.__modell
    def set_modell(self, value): # Setter
        self.__modell = value
```

Konstruktoren und Destruktoren

Ein **Konstruktor** ist die Methode einer Klasse, die automatisch beim Erzeugen eines Objektes aufgerufen wird. Konstruktoren werden meistens dafür benutzt, um das Objekt zu initialisieren, d.h. den Variablen Anfangswerte zuzuweisen oder weiteren Speicherplatz zu reservieren. Der Name des Konstruktors ist in Python immer `__init__` und in der Klammer folgt eine Liste mit Übergabeparametern. Die Parameterliste enthält auf jeden Fall `self` als ersten Übergabeparameter.

Damit wird eine Referenz auf das zugehörige Objekt übergeben. Bei Bedarf können noch weitere Parameter folgen. Für die Parameterübergabe gelten die gleichen Regeln wie bei Methoden, d. h., durch die Vorbelegung der Parameter mit Standardwerten muss beim Aufruf nicht zwingend für alle Parameter ein Wert übergeben werden. Abgeschlossen wird die Definition des Konstruktors wie bei Methoden üblich mit einem Doppelpunkt. In Python ist der Konstruktor eine private Methode, auf welche nur beim Erzeugen des Objektes zugegriffen werden kann. Auch

besitzt jede Klasse nur einen Konstruktor. Dieses unterscheidet Python von anderen Programmiersprachen wie C++ oder Java, bei denen es auch mehrere Konstruktoren geben kann.

Ein Destruktor ist jene Methode einer Klasse, die als Letztes vor dem Zerstören des Objektes aufgerufen wird. Damit sind Destruktoren das Gegenteil von Konstruktoren. Auch hier kann es nur einen Destruktor pro Klasse geben. Der Name des Destruktors in Python ist immer `del` und in der Klammer steht die Referenz `self` als einziger Übergabeparameter. Wie beim Konstruktor ist auch die Sichtbarkeit des Destruktors *private* und die Definition wird mit einem Doppelpunkt abgeschlossen. In der Regel werden Destruktoren dafür benutzt, um bestimmte Aktionen (z. B. Sichern von Daten) vor dem Löschen des Objektes auszuführen. Das folgende Beispiel zeigt den Konstruktor der Klasse „Musikinstrument“ und einen Destruktor, welcher exemplarisch mit aufgenommen wurde.

```
</> #Konstruktor
def __init__(self,modell = "unbekannt",hersteller = "unbekannt",preis =0.0):
    self.__modell = modell
    self.__hersteller = hersteller
    self.__preis = preis

def __del__(self): # Destruktor, hier nur als Beispiel ohne Funktion
    pass
```

Die Umsetzung der kompletten Klasse „Musikinstrument“ in Python würde wie folgt aussehen. Die Datentypen (string, double), welche im UML-Diagramm hinter den Eigenschaften auftauchen, können in Python vernachlässigt werden.

Umsetzung der Beispielsklasse „Musikinstrument“

```
</> class Musikinstrument:
# -----
# Attribute der Klasse mit entsprechenden Zugriffsmodifiern
# -----
__modell = "unbekannt"
__hersteller = "unbekannt"
__preis = 0.0
# -----
# Konstruktors der Klasse
# -----
def __init__(self, modell = "unbekannt", hersteller = "unbekannt",
preis = 0.0):
    self.__modell = modell
    self.__hersteller = hersteller
    self.__preis = preis
# -----
# Getter und Setter der Klasse
# -----
def get_modell(self):
    return self.__modell
def set_modell(self, value):
    self.__modell = value
def get_hersteller(self):
    return self.__hersteller
def set_hersteller(self, value):
    self.__hersteller = value
def get_preis(self):
    return self.__preis
def set_preis(self, value):
    self.__preis = value
# -----
# Methoden
# -----
def get_daten(self):
    daten = "Modell: " + self.__modell + "\n" + "Hersteller: " + \
        self.__hersteller + "\n" + \
        "Preis: " + str(self.__preis)

    return daten
```

Dieser Teil kann auch weggelassen werden.

Objekte

Objekte werden in Python generell aus Klassen erzeugt. Dazu wird zunächst ein Bezeichner für das Objekt angegeben, wobei der Bezeichner wieder frei gewählt werden kann. Danach wird das Objekt erzeugt und die Referenz dem Bezeichner zugewiesen. Diese erfolgt dadurch, dass erst der Klassenname mit entsprechenden Übergabeparametern in Klammern aufgerufen wird. Die Übergabeparameter werden dem Konstruktor automatisch übergeben.

Allgemeine Syntax

```
</> Bezeichner = Klasse(Parameterliste)
```

Zur Veranschaulichung dient das Beispiel „Erzeugen eines Objektes“ aus der Klasse „Musikinstrument“. Über die Getter und Setter des Objektes kann nun auf die Attribute zugegriffen werden. In dem Beispiel werden im Hauptprogramm dem Objekt konkrete Werte zugewiesen und danach die Daten mithilfe der Methode „get_daten()“ angezeigt.

Quelltext des Hauptprogramms „Musikinstrument“

```
</> from musikinstrument import Musikinstrument

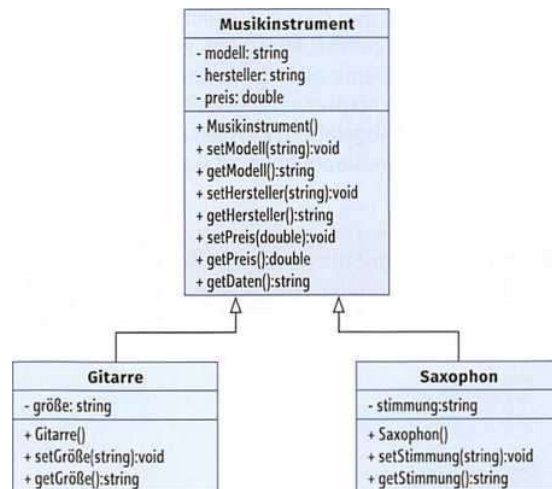
instrument =Musikinstrument("PRS Custom 24 BW", "PRS") # Objekt anlegen

def main():
    instrument.set_preis(3998.00)
    print(instrument.get_daten())

main() # Ausführen des Programms
```

LE 3 - Vererbung

Neben der Abstraktion und der Kapselung ist die Vererbung ein weiteres Basisprinzip der OOP. Über die Vererbung kann man eine neue Klasse auf Basis einer bereits vorhandenen Klasse erzeugen. Dabei wird zwischen Einfach- und Mehrfachvererbung unterschieden. Im Gegensatz zu anderen Programmiersprachen erlaubt Python Einfach- und Mehrfachvererbung-Vererbung wird in Python dadurch gekennzeichnet, dass nach dem Klassennamen in runden Klammern angegeben wird, von welcher Klasse die aktuelle Klasse erbt. Als Beispiel soll die Klasse „Musikinstrument“ um die Klassen „Gitarre“ und „Saxophon“ erweitert werden. Diese Erweiterung ist im UML-Klassendiagramm dargestellt. Dabei erben die beiden neuen Klassen die Attribute und Methoden der Klasse „Musikinstrument“. Es kann aber nur auf die Methoden zugegriffen werden, da diese *public* sind. Die Eigenschaften sind als *private* gekennzeichnet und deswegen wird der Zugriff auf diese Methoden in den abgeleiteten Klassen verweigert. Soll der Zugriff möglich sein, müssen die Attribute das Zugriffsrecht *protected* oder *public* erhalten. Die Umsetzung der Vererbung ist in dem nachstehenden Quellcode ersichtlich. Hier wird aber nur die Implementierung der beiden neuen Klassen aufgeführt.



Umsetzung der Beispielsklassen „Gitarre“ und „Saxophon“

```
</> class Gitarre(Musikinstrument):
    def __init__(self, modell = "unbekannt", hersteller = "unbekannt",
                  preis = 0.0, gröÙe = "unbekannt"):
        super().__init__(modell, hersteller, preis)
        self.__gröÙe = gröÙe
    def get_gröÙe(self):
        return self.__gröÙe
    def set_gröÙe(self, value):
        self.__gröÙe = value

class Saxophon(Musikinstrument):
    def __init__(self, modell = "unbekannt", hersteller = "unbekannt",
                  preis = 0.0, stimmung = "unbekannt"):
        super().__init__(modell, hersteller, preis)
        self.__stimmung = stimmung
    def get_stimmung(self):
        return self.__stimmung
    def set_stimmung(self, value):
        self.__stimmung = value
```

LE 4 - Überladen und Überschreiben von Methoden

Das Überladen von Methoden ist, im Gegensatz zu anderen Programmiersprachen, in Python nicht möglich. Wenn eine Methode mehrfach definiert wird, dann gilt immer die letzte Definition dieser Methode. Dieses gilt auch, wenn die Methoden unterschiedliche Übergabeparameter besitzen. Methoden können aber in abgeleiteten Klassen überschrieben werden.

Das Überschreiben von Methoden bedeutet, dass eine Methode einer Basisklasse in der abgeleiteten Klasse neu implementiert (überschrieben) wird. Diese Vorgehensweise wird an dem Beispiel der Methode „get_daten()“ aus der Klasse „Musikinstrument“ illustriert. Diese Methode wird von den Klassen „Gitarre“ und „Saxophon“ geerbt. Allerdings gibt sie bisher nur die Werte für die Attribute „modell“, „hersteller“ und „preis“ aus. Damit alle Werte des entsprechenden Objektes ausgegeben werden, muss der Quelltext dieser Methode entsprechend erweitert werden. Dieses kann aber nicht in der Klasse „Musikinstrument“ geschehen, weil abgeleitete Klassen unterschiedliche Attribute besitzen. Also muss diese Methode in den neuen Klassen neu implementiert werden und damit wird gleichzeitig die geerbte Methode „get_daten()“ überschrieben.

Damit direkt auf die Attribute der Klasse „Musikinstrument“ zugegriffen werden kann, müssen dort die Zugriffsmodifizier von *private* auf *protected* (public) verändert werden. Alternativ könnten auch die entsprechenden Getter und Setter verwendet werden.

Überschreiben der Methode „get_daten()“ in den Beispielsklassen „Gitarre“ und „Saxophon“

```
</> class Gitarre(Musikinstrument):
    ...
    def get_daten(self):
        daten = "Modell: " + self._modell + "\n" + \
            "Hersteller: " + self._hersteller + "\n" + \
            "Größe: " + self._größe + "\n" + "Preis: " + \
            str(self._preis)
        return daten
    ...

class Saxophon(Musikinstrument):
    ...
    def get_daten(self):
        daten = "Modell: " + self._modell + "\n" + \
            "Hersteller: " + self._hersteller + "\n" + \
            "Stimmung: " + self._stimmung + "\n" + "Preis: " + \
            str(self._preis)
        return daten
    ...
```

LE 5 - Abstrakte Klasse und Interface

Abstrakte Klasse

Eine abstrakte Klasse ist eine spezielle Klasse, die neben normalen Methoden auch sogenannte abstrakte Methoden enthält. Abstrakte Methoden sind Methoden, die eine Methodensignatur, aber keinen Methodenrumpf besitzen. Abstrakte Methoden sind nicht implementiert und können deshalb auch nicht ausgeführt werden. Aus diesem Grund ist es auch nicht möglich, von abstrakten Klassen Objekte zu bilden. Oder anders ausgedrückt: Es ist nicht möglich, abstrakte Klassen zu instanziiieren. Sinn und Zweck einer abstrakten Klasse bestehen darin, dass von ihr geerbt wird. Die nicht implementierten Methoden müssen dann in der neuen Klasse „ausprogrammiert“ also überschrieben werden. Erst dann ist es möglich, eine Instanz von der neuen Klasse zu bilden. Verwendet werden abstrakte Klassen beispielsweise als Basisklassen, um grundlegende Eigenschaften und Methoden ihrer Unterklassen festzulegen, ohne diese bereits konkret zu implementieren. Ab Python 3.4 kann man eine abstrakte Klasse erstellen, indem man von „ABC“ erbt. Dieses Modul muss vorher mit folgender Zeile ins Programm eingebunden werden: `from abc import ABC, abstractmethod`.

In einer abstrakten Klasse können beliebig viele abstrakte Methoden deklariert werden, die durch `@abstractmethod` gekennzeichnet werden. Abstrakte Methoden haben keine Implementierung und enthalten nur das Schlüsselwort „pass“. Zur Veranschaulichung soll das folgende Beispiel dienen.

Abstrakte Klasse „Musikinstrument“

```
</> from abc import ABC, abstractmethod
class Musikinstrument(ABC):
    ...
    @abstractmethod
    def get_daten(self):
        pass
```

Der Rest der Klasse bleibt gleich.

In dem Beispiel wird aus der normalen Klasse „Musikinstrument“ eine abstrakte Klasse, indem die Methode „get_daten()“ in eine abstrakte Methode umgewandelt wird. Dazu wird dieses als abstrakt gekennzeichnet und kein Methodenrumpf implementiert wird. Der Rest des Programms bleibt gleich. Da nun von der Klasse „Musikinstrument“ kein Objekt mehr angelegt werden kann, muss das Hauptprogramm entsprechend angepasst werden. Dazu werden jeweils eine Gitarre und ein Saxophon erzeugt und mit Daten versehen, welche im Anschluss wieder ausgegeben werden.

Angepasstes Hauptprogramm

```
</> from gitarre import Gitarre
from saxophon import Saxophon

gitarre = Gitarre("PRS Custom 24 BW", "PRS", 3998.00) # Objekt anlegen
saxophon = Saxophon("Yamaha YAS-280S", "Yamaha", 1179.00) # Objekt anlegen

def main():
    gitarre.set_größe("4/4")
    print(gitarre.get_daten())
    saxophon.set_stimmung("Eb")
    print(saxophon.get_daten())

main() # Ausführen des Programms
```

Interface

In Python gibt es kein entsprechendes Schlüsselwort und damit auch kein Interface. Ein Interface kann in Python aber über eine abstrakte Klasse gestaltet werden. In dieser Klasse sind dann alle Methoden abstrakt.

LE 6 - Weitere Beispielaufgabe

Die nachfolgende Beispielaufgabe wurde in ihrer Komplexität reduziert. Weiterhin wurde im Klassendiagramm auf die Angabe der Getter- und Setter-Methoden und des Konstruktors verzichtet. Diese sind im Programm trotzdem zu implementieren. Bei der Lösung wird auch nur die Implementierung der Klassen dargestellt. Ein entsprechendes Hauptprogramm muss noch entwickelt werden.

Beispielaufgabe

In der Planung für ein Python-Programm zur Gebäudeverwaltung wurde ein erstes Klassendiagramm entwickelt. Implementieren Sie dieses Klassendiagramm in Python. Geben Sie dabei alle Getter- und Setter-Methoden an. Außerdem soll jede Klasse einen Konstruktor erhalten, in dem die Eigenschaften initialisiert werden.



Lösung

Zunächst wird die Klasse „Gebäude“ implementiert. Da die Eigenschaften der Klasse *private* sein sollen, müssen unbedingt auch die dazugehörigen öffentlichen Getter und Setter entwickelt werden, damit auf diese auch in den abgeleiteten Klassen zugegriffen werden kann.

```
</> class Gebäude:
def __init__(self, adresse = "unbekannt", preis = 0.0, baujahr = 0):
    self.__adresse = adresse
    self.__preis = preis
    self.__baujahr = baujahr

def get_adresse(self):
    return self.__adresse
def set_adresse(self, value):
    self.__adresse = value
def get_preis(self):
    return self.__preis
def set_preis(self, value):
    self.__preis = value
def get_baujahr(self):
    return self.__baujahr
def set_baujahr(self, value):
    self.__baujahr = value

def get_daten(self):
    daten = "Adresse: " + self.__adresse + "\n" + \
           "Preis: " + str(self.__preis) + "\n" + \
           "Baujahr: " + str(self.__baujahr)
    return daten
```

Aus dieser Klasse werden nun die Klassen „Wohnhaus“ und „Lagerhalle“ abgeleitet. Die Implementierung dieser beiden Klassen sieht wie folgt aus:

```
</> class Wohnhaus(Gebäude):
def __init__(self, adresse = "unbekannt", preis = 0.0,
    baujahr = 0, anzahlWohnungen = 0):
    super().__init__(adresse, preis, baujahr)
    self.__anzahlWohnungen = anzahlWohnungen
def get_anzahlWohnungen(self):
    return self.__anzahlWohnungen
def set_anzahlWohnungen(self, value):
    self.__anzahlWohnungen = value

class Lagerhalle(Gebäude):
def __init__(self, adresse = "unbekannt", preis = 0.0,
    baujahr = 0, anzahlWohnungen = 0):
    super().__init__(adresse, preis, baujahr)
    self.__lagerfläche = lagerfläche
def get_lagerfläche(self):
    return self.__lagerfläche
def set_lagerfläche(self, value):
    self.__lagerfläche = value
```

TAG 21 - Grafische Benutzerschnittstellen in Python entwickeln

Die Programmierung einer grafischen Benutzeroberfläche in Python ist sehr komplex und wird daher nur angerissen. Der wesentliche Unterschied zwischen grafischer Benutzeroberfläche und Konsolenanwendung besteht in der unterschiedlichen Interaktion des Benutzers mit dem Programm, d. h., es sind z. B. die Ein- und Ausgabe verschieden. Eine grafische Benutzeroberfläche bietet hier viel mehr Möglichkeiten



der Interaktion, nicht zuletzt durch eine sehr große Anzahl an Oberflächenelementen. Das Entwerfen und Gestalten von Oberflächen, die erwartungskonform und benutzerfreundlich sind, ist vielfach mit erheblichem Aufwand verbunden. Der folgende Abschnitt beschränkt sich darauf, die Grundlagen anhand von reduzierten Beispielen zu erläutern.

LE 1 - Grundlagen

Um eine grafische Benutzeroberfläche unter Python zu erstellen, wird ein sogenanntes Toolkit (grafische Bibliothek) benötigt. Beispiele für derartige Bibliotheken sind etwa Microsoft Foundation Classes (MFC) für Windows oder auch plattformunabhängige Bibliotheken wie Qt, Gtk usw. Wenn sogenannte Bindings für Python existieren, können diese damit in ein Python-Programm eingebunden werden. Für die Beispiele in diesem Buch wird Tkinter (Tk) verwendet. Dieses Toolkit ist in der Standardbibliothek von Python enthalten und muss nicht extra installiert werden. Damit ist es möglich, kleine GUI-Programme zu schreiben, und es stellt für die Einführung in die Oberflächenprogrammierung das ideale Werkzeug dar. In der folgenden Tabelle sind noch weitere Toolkits aufgeführt, die Bindings für Python bereitstellen. Allerdings stammen sie von anderen Anbietern und müssen extra installiert werden.

Toolkit	Beschreibung
PyObject	Diese Bibliothek stellt ein Binding für das Gtk-Toolkit bereit. Gtk zählt neben Qt zu den verbreitetsten plattformunabhängigen Toolkits.
PyQt	Diese Bibliothek stellt ein Binding für das Qt-Toolkit bereit. Bei Qt handelt es sich um ein sehr umfassendes Framework, es ist ähnlich verbreitet wie Gtk.

Der grundlegendste Bestandteil einer grafischen Benutzeroberfläche ist das Fenster. Daher wird zunächst kurz darauf eingegangen, wie solch ein Fenster mit einem Button zum Schließen der Anwendung unter Python erzeugt und angezeigt wird. Zunächst wird wieder ein Python-Projekt angelegt und gespeichert. Danach werden mit folgendem Quellcode ein Fenster und ein Button erzeugt:

Beispiel für ein Fenster mit Button in Python

```

</> import tkinter
      from tkinter import ttk

      def btnClose_click():
          frmMain.destroy()

      # -----
      # Erzeugen des Hauptfensters
      # -----
      frmMain = tkinter.Tk()
      frmMain.title("Ein Fenster mit Button")
      frmMain.wm_geometry('400x200')

      # -----
      # Erzeugen und initialisieren des Button "btnClose"
      # -----
      btnClose = ttk.Button(frmMain, text = "Beenden", command = btnClose_click)
      btnClose.place(x=160,y=80)

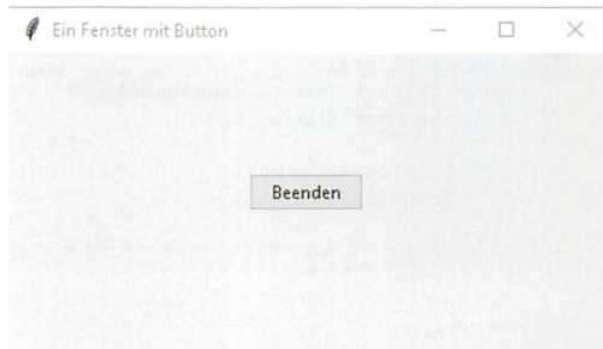
      # -----
      # Endlosschleife
      # -----
      frmMain.mainloop()
  
```

Importieren der notwendigen Module

Mit diesem Befehl wird die Anwendung geschlossen.

Setzen des Fenstertitels und der Fenstergröße

In der ersten Zeile des Programms wird das Modul „tkinter“ importiert. Dieses Modul enthält alle Klassen, die notwendig sind, um die Benutzeroberfläche zu erstellen. Im folgenden Teil des Programms wird ein Objekt der Klasse „tk.Tk“ erzeugt, welches den Namen „frmMain“ erhält. Über den Objektnamen und die entsprechende Eigenschaft werden einzelne Werte des Fensters gesetzt. Um Ereignisse für das Fenster abzufangen und auszuwerten, ist außerdem der Aufruf der Funktion „mainloop()“ des Fensters notwendig. Dieses erfolgt mit der Zeile „frmMain.mainloop()“ am Ende des Programms. Damit ist das Grundgerüst einer GUI-Anwendung in Python fertig.



Um einen Button anzuzeigen, muss zunächst ein Objekt der Klasse „Button“ erstellt werden. In dem Beispiel wird dem Button beim Erzeugen sein Elternfenster „frmMain“, sein Text „Beenden“ und die Methode „btnClose_click“ zugewiesen. Danach wird er auf der Oberfläche angeordnet. Nach dem Starten des Programms ist das abgebildete Fenster zu sehen. Das Fenster selbst ist unspektakulär. Es besitzt aber bereits die Grundfunktionalität eines jeden Fensters. Es kann z. B. vergrößert und verkleinert, verschoben oder geschlossen werden.

LE 2 - Weitere Beispielaufgabe

Anhand eines Beispiels wird nun gezeigt, wie ein Programm mit einer grafischen Benutzeroberfläche unter Python realisiert werden kann.

Beispielaufgabe

Es soll ein Programm mit einer grafischen Benutzerschnittstelle entwickelt werden, welches nach der Eingabe der Tankmenge und des Preises pro Liter die Tankkosten ausrechnet und diese ausgibt.

Lösung

Zunächst werden die Oberfläche und der Algorithmus geplant. Für diese einfache Aufgabe könnte man diese Phase überspringen und gleich zur Umsetzung in Python übergehen. Allerdings soll hier an einem einfachen Beispiel der vollständige Entwicklungsprozess aufgezeigt werden, ohne ins Detail zu gehen.

Mockup der Oberfläche

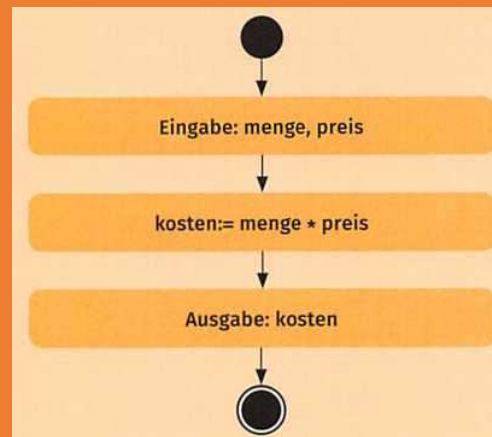
Kostenrechner: Tanken

Menge in Liter

Preis pro Liter in Euro

Kosten inEuro

Algorithmus als Aktivitätsdiagramm



Nach der Planung erfolgt nun die Umsetzung in Python. Dazu werden im Quelltext ein entsprechendes Fenster und die notwendigen Widgets implementiert. Eine mögliche Lösung könnte wie folgt aussehen:

```

</> import tkinter
from tkinter import messagebox
from tkinter import ttk

def btnClose_click():
    frmMain.destroy()
  
```

Import der benötigten Module für Message-Boxen und die Widgets

Anwendung schließen

```
def btnCalc_click():
    try:
        menge = float(txtVolume.get())
        preis = float(txtPrice.get())
        ergebnis = menge * preis
        txtResult.delete(0, 'end')
        txtResult.insert(0, ergebnis)
    except Exception as e:
        messagebox.showwarning("Warnung",
                               "Es ist folgender Fehler aufgetreten: \n"
                               + e.args[0])

# Erzeugen des Hauptfensters
frmMain = tkinter.Tk()

frmMain.title("Kostenrechner: Tanken")
frmMain.wm_geometry('350x200')

# Anlegen der Buttons
btnClose = ttk.Button(frmMain, text = "Beenden", command = btnClose_click)
btnClose.place(x=220, y=150)

btnCalc = ttk.Button(frmMain, text = "Berechnen", command = btnCalc_click)
btnCalc.place(x=120, y=150)

# Anlegen der Labels
lblVolume = ttk.Label(frmMain, text = "Menge in Liter")
lblVolume.place(x=50, y=50)

lblPrice = ttk.Label(frmMain, text = "Preis pro Liter in Euro")
lblPrice.place(x=10, y=80)

lblResult = ttk.Label(frmMain, text = "Kosten in Euro")
lblResult.place(x=50, y=110)

# Anlegen der Entries
txtVolume = ttk.Entry(frmMain, foreground = "red")
txtVolume.insert(0, "0")
txtVolume.place(x=140, y=50)

txtPrice = ttk.Entry(frmMain, foreground = "red")
txtPrice.insert(0, "0")
txtPrice.place(x=140, y=80)

txtResult = ttk.Entry(frmMain, foreground = "green")
txtResult.insert(0, "0")
txtResult.place(x=140, y=110)

# Endlosschleife
frmMain.mainloop()
```

Berechnung der Tankkosten nach dem geplanten Algorithmus und Ausgabe des Ergebnisses

Ab hier werden das Hauptfenster die Steuerelemente erzeugt und initialisiert.

Nachdem alles korrekt implementiert wurde, kann das Programm gestartet und getestet werden.

Beispielausgabe:



The screenshot shows a Java Swing window titled "Kostenrechner: Tanken". Inside the window, there are three text input fields with labels to their left: "Menge in Liter" with the value "55", "Preis pro Liter in Euro" with the value "1.45", and "Kosten in Euro" with the value "79.75". The values "55" and "1.45" are highlighted in red. Below the input fields are two buttons: "Berechnen" and "Beenden".

Label	Value
Menge in Liter	55
Preis pro Liter in Euro	1.45
Kosten in Euro	79.75

TAG 22 - Datenbanklösungen bedarfsgerecht entwickeln

Um Daten systemübergreifend bereitzustellen, müssen entsprechende Datenbanklösungen entwickelt werden. Dazu stehen heutzutage immer mehr Möglichkeiten zur Verfügung. Sie können aus unterschiedlichen Datenmodellen und Datenbankmanagementsystemen sowie dem Ort der Speicherung der Daten, z. B. in der Cloud, auswählen. Allerdings passt nicht jeder Lösungsansatz zu jedem Problem. Deswegen müssen Datenbanklösungen sorgfältig geplant und umgesetzt werden. Der dazu notwendige Prozess wird nachfolgend beschrieben. Weiterhin wird vorhandenes Wissen bezüglich relationaler Datenbanken und SQL erweitert und vertieft.



LE 1 - Ein geeignetes Datenmodell auswählen

Daten können in strukturierte, semistrukturierte und unstrukturierte Daten eingeteilt werden. Bis in die späten 2000er-Jahre standen die Speicherung und Verarbeitung von strukturierten Daten im Mittelpunkt der Anwendungsentwicklung. Unstrukturierte Daten spielten damals noch keine große Rolle. Dementsprechend hoch war der Stellenwert von Datenbanklösungen, welche auf dem relationalen Datenbankmodell beruhen.

Beispiele für unstrukturierte Daten				
Text	Grafik	Bild	Audio	Video
• Fließtext • Strukturierter Text • Textsammlung • usw.	• Landkarte • 3D-Modelle • Technische Zeichnung • usw.	• Fotos • Röntgenbilder • Satellitenbilder • usw.	• Musik • Sprache • Geräusche • usw.	• Film • Videokonferenz • Animation • usw.

Andere Ansätze wie objektorientierte Datenbanken erreichten aufgrund ihrer komplexen Handhabung und Implementierung nie diesen hohen Stellenwert. Dieses Szenario änderte sich langsam mit dem Aufkommen von neuen IT-Technologien. Diese sorgten für eine stetige Zunahme der Datenmenge, wobei neben strukturierten Daten vor allem die Menge an unstrukturierten Daten wie Fotos, Videos oder Textnachrichten zunahm. Dieser Trend setzt sich bis heute fort. Aktuell sorgen vor allem IoT, künstliche Intelligenz, 5G-Mobilfunk, Edge Computing und neue, datenbasierte Geschäftsmodelle für ein anhaltendes Datenwachstum. In diesem Zusammenhang fällt auch immer mehr der Begriff „Big Data“. Big Data bezeichnet in erster Linie die Verarbeitung großer, komplexer und sich schnell ändernder Datenmengen.

Die fünf Big Data Vs	
Volume	Durch das Volumen wird die immer größere Menge an Daten beschrieben, welche zu speichern und zu verarbeiten ist.
Variety	Hierdurch wird die Vielfalt der Daten beschrieben. Rund 90% der gespeicherten Daten sind unstrukturierten Formaten wie Texten, Bildern oder Videos zuzuordnen. Durch Big Data werden diese Daten anhand von Machine Learning analysierbar.
Velocity	Hiermit ist die erhöhte Geschwindigkeit gemeint, mit der Daten produziert und verarbeitet werden müssen. Für viele Anwendungsfälle spielt die Echtzeitverarbeitung der Daten eine große Rolle und kann für den entscheidenden Wettbewerbsvorteil sorgen.
Veracity	Hier werden die Vertrauenswürdigkeit und die Qualität der Daten betrachtet, welche aus vielen sehr unterschiedlichen Quellen zusammengeführt werden.
Value	Durch den Value wird der Mehrwert für das Unternehmen durch die Auswertung der riesigen Datenmengen beschrieben. Er stellt für die Unternehmen, welche sich im Big- Data-Umfeld bewegen, den wichtigsten Punkt dar.

Diese Entwicklung hat die Anforderungen an Datenbanken drastisch verändert. Agilität, Flexibilität und Skalierbarkeit stehen zunehmend im Fokus der Datenbankentwicklung. Diese neuen Anforderungen können nur bedingt von den klassischen relationalen Datenbanken abgedeckt werden. Deswegen wurden neue Ansätze in Form der NoSQL-Datenbanken entwickelt, welche diesen Anforderungen besser gerecht werden. Durch die Flexibilität der NoSQL-Datenbanken können praktisch alle Arten von Daten gespeichert und verarbeitet werden, egal ob strukturiert, semi- oder unstrukturiert. Des Weiteren können solche Speicherlösungen unbegrenzt erweitert (skaliert) werden. An dieser Stelle stellt sich nun die Frage, wann relationale und wann NoSQL-Datenbankenlösungen bevorzugt werden sollten. Dazu werden die beiden Ansätze in der nachfolgenden Tabelle gegenübergestellt.

	Relationale Datenbank	NoSQL-Datenbank																																								
Datenmodell	Relationales Datenmodell; Daten werden in separaten Tabellen gespeichert und bei komplexen Suchanfragen vom System zusammengeführt. <table><thead><tr><th>ID</th><th>Nachname</th><th>Vorname</th><th>AdressID</th></tr></thead><tbody><tr><td>1</td><td>Maier</td><td>Frank</td><td>2</td></tr><tr><td>2</td><td>Müller</td><td>Ruth</td><td>3</td></tr><tr><td>3</td><td>Schulzes</td><td>Marion</td><td>7</td></tr><tr><td>...</td><td>...</td><td>...</td><td>...</td></tr></tbody></table> <table><thead><tr><th>AdressID</th><th>PLZ</th><th>Ort</th><th>Straße</th></tr></thead><tbody><tr><td>1</td><td>20355</td><td>Hamburg</td><td>...</td></tr><tr><td>2</td><td>10555</td><td>Berlin</td><td>...</td></tr><tr><td>3</td><td>80686</td><td>München</td><td>...</td></tr><tr><td>...</td><td>...</td><td>...</td><td>...</td></tr></tbody></table>	ID	Nachname	Vorname	AdressID	1	Maier	Frank	2	2	Müller	Ruth	3	3	Schulzes	Marion	7	...	...	...	...	AdressID	PLZ	Ort	Straße	1	20355	Hamburg	...	2	10555	Berlin	...	3	80686	München	...	...	...	...	...	Verschiedene Datenbankmodelle, z. B. Dokumenten-, Graph- oder Key-Value-Datenmodell. Key-Value Document Graph
ID	Nachname	Vorname	AdressID																																							
1	Maier	Frank	2																																							
2	Müller	Ruth	3																																							
3	Schulzes	Marion	7																																							
...	...	...	...																																							
AdressID	PLZ	Ort	Straße																																							
1	20355	Hamburg	...																																							
2	10555	Berlin	...																																							
3	80686	München	...																																							
...	...	...	...																																							
Schemata	Datentyp und -Struktur sind festgelegt. Um neue Attribute hinzuzufügen, muss teilweise die gesamte Datenbank angepasst werden.	Flexibel; strukturierte, semistrukturierte und unstrukturierte Daten können zusammen abgespeichert werden, eine vorherige Konvertierung ist nicht notwendig.																																								
Skalierung	Vertikale Skalierung; ein einzelner Server muss die Leistung des kompletten Datenbanksystems tragen. Die Leistungsfähigkeit des Servers kann bedingt gesteigert werden.	Horizontale Skalierung; durch Hinzufügen von neuen Datenbankservern kann das Datenbanksystem fast beliebig skaliert werden.																																								
Umsetzung der ACID-Regeln	Bei SQL-Datenbanken werden alle ACID-Regeln wie Atomicity, Consistency, Isolation, Durability umgesetzt.	Die ACID-Regeln werden meistens nicht unterstützt. Stattdessen wird das BASE-Modell verwendet (Basically Available, Soft State, Eventually Consistent). Es gilt: Verfügbarkeit vor Konsistenz.																																								
Leistung	Um die Leistung zu erhöhen, müssen Abfragen, Indizes und Struktur optimiert werden.	Durch das Nutzen von Cloud-Servern und Hardware-Clustern haben NoSQL-Datenbanken eine deutlich höhere Leistungsstärke.																																								
Abfragesprachen	Einsatz von SQL (Structured Query Language)	keine einheitliche Abfragesprache																																								
Populäre Datenbankmanagementsysteme	MySQL, PostgreSQL, Oracle, MS-SQL	MongoDB, Apache HBase, Amazon DynamoDB, Redis, Couchbase, Cassandra, Elasticsearch																																								
Einsatzgebiete	Datenbanklösungen, welche einen ACID-Support erfordern und bei denen sich die Struktur nicht wesentlich verändert oder die Datenmenge unkontrolliert wächst. Vor allem für Daten, welche sich leicht strukturieren lassen.	Datenbanklösungen, welche - eine Echtzeitverarbeitung von Daten erfordern, - Daten verarbeiten, die sich schlecht strukturieren lassen, - große, schnell wachsende Datenmengen zu bewältigen haben.																																								

Je nach Anwendungsgebiet kann eine NoSQL-Datenbanklösung entscheidende Vorteile gegenüber klassischen relationalen Datenbanklösungen bieten. In Bereichen, in denen relationale Datenbanken schnell überlastet sind, z. B. im Big-Data-Umfeld, warten NoSQL-Datenbanken mit hochleistungsfähigen, skalierbaren Datenmodellen auf, die große Datenmengen mit hoher

Geschwindigkeit verarbeiten können. Dabei sind diese Datenmodelle sehr flexibel, und dank der Speicherung der Daten auf verteilten Hardware- Clustern sind diese Datenbanklösungen weniger anfällig für Störungen und deutlich preiswerter.

ACID-Modell			
Atomicity (Atomarität)	Consistency (Konsistenz)	Isolation (Abgrenzung)	Durability (Dauerhaftigkeit)
Alle Aktionen einer Transaktion werden komplett oder gar nicht ausgeführt.	Ist eine Transaktion erfolgreich abgeschlossen, muss sie in der zuvor konsistenten Datenbank einen wieder konsistenten Zustand hinterlassen.	Stellt sicher, dass die parallele Nutzung der Datenbank durch mehrere Anwender keine negativen Auswirkungen nach sich zieht.	Ist eine Transaktion ausgeführt und konsistent, sind ihre Informationen dauerhaft in der Datenbank gespeichert.

Die Vorteile von NoSQL-Datenbanklösungen haben jedoch ihren Preis. Da die NoSQL-Datenbanken anstatt dem ACID-Modell (Atomicity, Consistency, Isolation und Durability) dem BASE-Modell (Basic Availability, Soft State, Eventual Consistency) folgen, sind eine unmittelbare Konsistenz und Zuverlässigkeit der Daten nicht gegeben. Außerdem fehlen Mechanismen zur Überprüfung der Datenintegrität. Dies muss im externen Code erfolgen. Des Weiteren werden komplexe SQL-Operationen, z. B. Tabellen-Joins, nicht unterstützt.

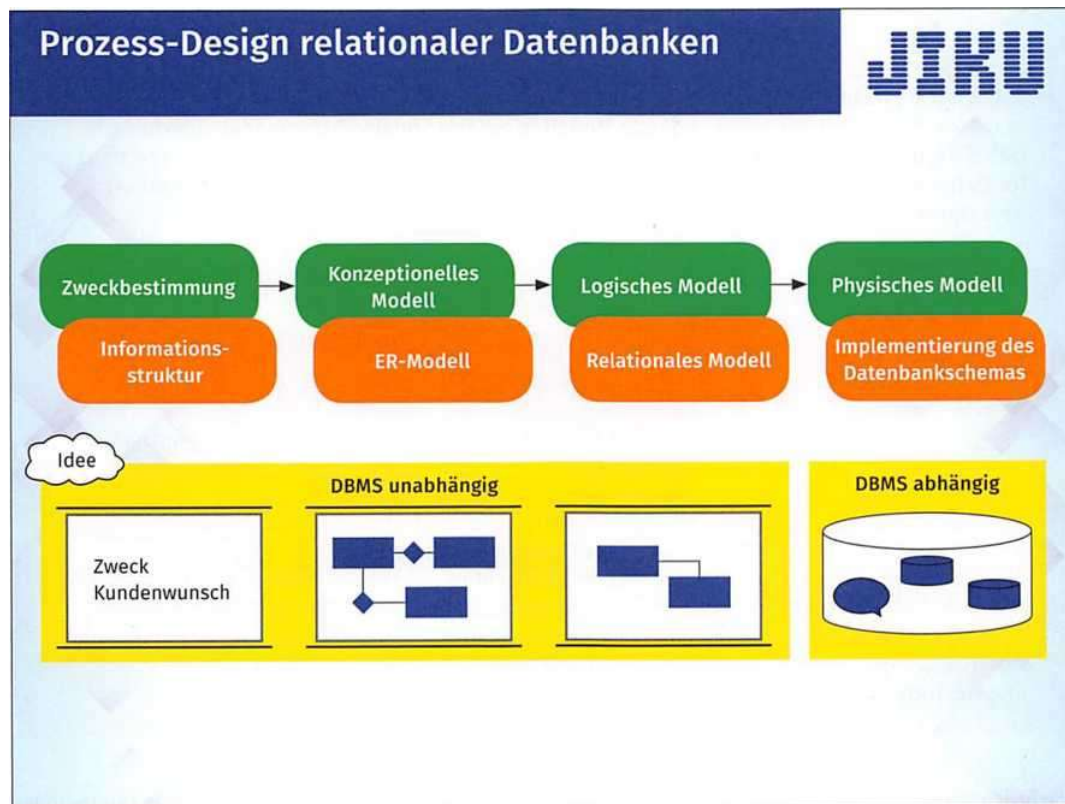
BASE-Modell (Basically Available, Soft State, Eventual Consistency)

Die Bezeichnung BASE ist kein Initialwort, sondern als Kurzwort entstanden, um einen sprachlichen Gegensatz zwischen BASE (Base) und ACID (Säure) zu schaffen. Es gibt daher keine präzise Definition für „basically available“ oder „soft state“. Allerdings wird die absolute Konsistenz der Daten aufgegeben und stattdessen deren Verfügbarkeit erhöht. Dadurch kann zeitweise auch ein inkonsistenter Zustand auftreten.

NoSQL-Datenbanken sind auf bestimmte Anwendungsgebiete spezialisiert. Hierfunktionieren sie sehr gut. Für andere Anwendungsgebiete bieten sie allerdings eine schlechte Performance. In diesem Umfeld haben sich vor allem die relationalen Datenbanken über Jahre hinweg bewährt und sind auch in den Zeiten von NoSQL-Datenbanken und der Cloud unverzichtbar. Sie kommen vor allem zum Einsatz, wenn ein komplexes unveränderliches Schema mit vielen Entitäten und Beziehungen gefordert ist und, wenn es um die Einhaltung der ACID-Regel geht. Dieses ist beispielsweise in allen Bereichen erforderlich, in denen es um das Thema „Geld“ geht. Alle diese Punkte müssen im Vorfeld sorgfältig abgewogen werden. Je nachdem, für welche Datenbanklösung man sich entscheidet, fällt der Prozess des Datenbankdesigns danach etwas anders aus. Im nächsten Abschnitt wird der Designprozess von relationalen Datenbanken näher betrachtet.

LE 2 - Den Prozess des Designs relationaler Datenbanken beschreiben

Relationale Datenbanken sind heutzutage für viele Unternehmen und Behörden ein zentraler und unersetzlicher Bestandteil bei der Speicherung, Verwaltung und Auswertung großer Datenmengen. Deswegen besteht immer noch ein sehr hoher Bedarf an entsprechenden relationalen Datenbanklösungen, welche eine hohe Qualität der darin gespeicherten Daten aufweisen. Dem entsprechend müssen relationale Datenbanken sorgfältig entworfen und umgesetzt werden. In der Regel durchläuft der Designprozess die nachfolgenden vier Phasen.



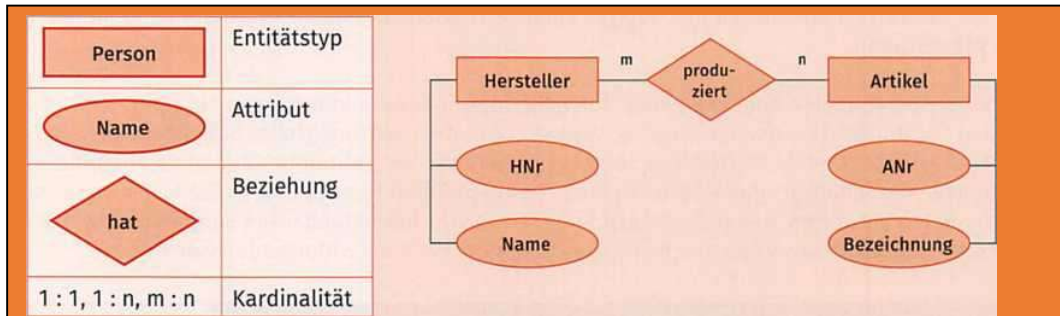
Phasen des Designprozesses von relationalen Datenbanken

1. Analysephase (Ermittlung der Informationsstruktur)

Zunächst sind in einer Anforderungsanalyse die Anforderungen des Kunden an das zu entwickelnde Datenbanksystem zu ermitteln und zu strukturieren. Das Ergebnis dieser ersten Phase ist eine informelle Beschreibung des zu lösenden Problems.

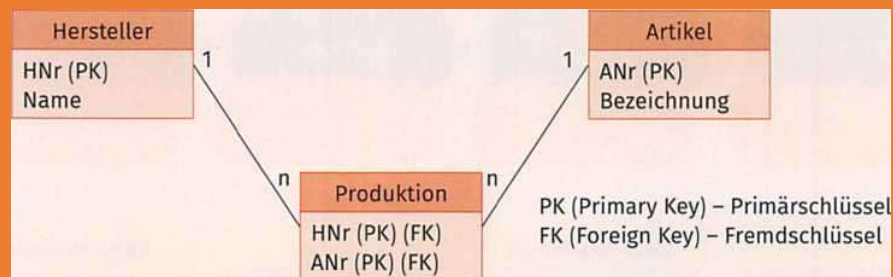
2. Konzeptionelle Phase (Entwicklung eines konzeptionellen Modells)

Das Ziel der konzeptionellen Phase ist eine formalisierte Beschreibung des betrachteten Sachverhalts. Dazu gibt es mehrere Möglichkeiten. Der bekannteste Ansatz ist das ER-Modell.



3. Logische Phase (Erstellung eines logischen Datenmodells)

In dieser Phase wird das konzeptionelle Modell in ein relationales Datenmodell überführt. Dabei werden zwei Schritte durchlaufen. Im ersten Schritt wird das ER-Modell in ein relationales Datenmodell überführt. Im zweiten Schritt erfolgt eine Optimierung des relationalen Datenmodells durch Normalisierung.



4. Implementationsphase (Erstellung der physischen Datenbank)

Am Ende dieser Phase sollte eine leere funktionstüchtige Datenbank existieren. Dazu wird das logische Modell mithilfe von SQL in ein konkretes Datenbankschema übersetzt. Im Zuge dessen müssen Datentypen, Wertebereiche, Relationen usw. festgelegt werden.

CREATE TABLE Hersteller (HNr INTEGER, Name VARCHAR(50), PRIMARY KEY(HNr))

Natürlich kann man bei einfachen Datenstrukturen die konzeptionelle und logische Phase zusammenfassen und schnell zur technischen Umsetzung der Datenbank voranschreiten. Dabei sollte jedoch beachtet werden, dass spätere Änderungen in der Struktur von relationalen Datenbanken nur mit viel Aufwand vorgenommen werden können. Deswegen sollten Abkürzungen in der Designphase sorgfältig abgewogen werden. Zumindest das logische Design sollte stimmig sein. Ein schlechtes logisches Design und das daraus resultierende schlechte physikalische Design können unter Umständen eine Zeitlang durch leistungsfähige Hardware ausgeglichen werden. Aber am Ende führt es immer zu Problemen. Nachfolgend ein Beispiel, wie aus einem Kundenauftrag ein konzeptionelles und danach ein relationales Modell entwickelt wird. Die Normalisierung des relationalen Modells wird im nächsten Abschnitt thematisiert.

Beispiel für den Buchverleih einer Bibliothek

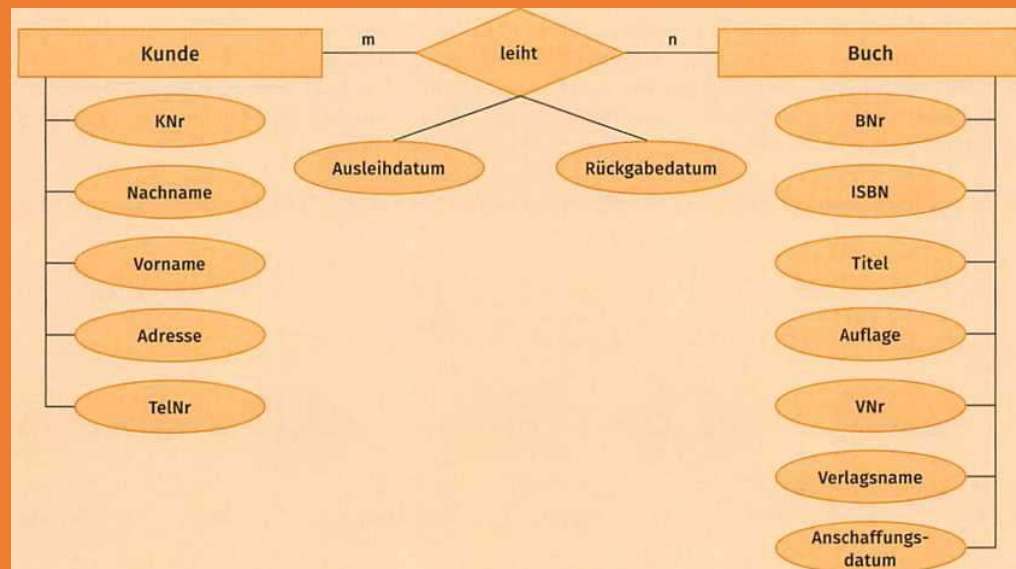
Analysephase (wurde für das Beispiel stark vereinfacht)

Es soll eine relationale Datenbank erstellt werden, mit deren Hilfe die Bücher einer Bibliothek und die Verleihungsvorgänge verwaltet werden. Folgende Daten soll die Datenbank enthalten:

- Kunde: Kundennummer, Nachname, Vorname, Adresse, Telefonnummer
- Buch: Titel, ISBN, Auflage, Verlag, Verlagsnummer, Buchnummer, Anschaffungsdatum
- Verleihvorgang: Buch, Kunde, Ausleihdatum, Rückgabedatum

Aus diesen Informationen wird zunächst ein ER-Modell entwickelt.

Konzeptionelle Phase



Aus dem ER-Modell wird nun ein relationales Modell abgeleitet, welches aber noch nicht normalisiert ist.

Logische Phase (Teil 1)



Damit ist der erste Teil der logischen Phase abgeschlossen. Im Anschluss daran wird in einem zweiten Schritt das relationale Modell normalisiert. Der Vorgang des Normalisierens wird im nächsten Abschnitt behandelt.

TAG 23 - Relationale Datenmodelle normalisieren

Die Einhaltung der Integrität bei einer relationalen Datenbank erzwingt bestimmte Anforderungen an die Form der Tabellen, die über den Weg der Normalisierung erreicht werden können. Am Ende der Normalisierung stehen Relationen, die einer vorgegebenen Normalform entsprechen. Je höher die gewählte Normalform (NF) definiert wird, desto höher sind die Anforderungen an die innere Struktur der Relation.

Normalisierung

Normalisierung ist ein Verfahren zur Verringerung von Datenredundanz in relationalen Datenbankmodellen verbunden mit dem Ziel, die Datenkonsistenz zu erhöhen.

Durch die Normalisierung sollen einerseits die Relationen aufgebaut und andererseits die Redundanzen verringert bzw. ganz vermieden werden. In der Praxis beschränkt man sich auf die erste bis dritte Normalform. Die vierte und fünfte Normalform werden nur in seltenen Fällen angewandt. Als Beispiel dient die folgende nicht normalisierte Tabelle „Bestellung“.

Beispieltabelle mit Datensätzen

Tabelle „Bestellung“

BestellNr	Datum	Kunde	Bestellpositionen
1	12.12.2020	Maier - KNr: 71	1. 2 Tische Nr. 12, 2. 3 Schränke Nr. 88
2	14.12.2020	Maier - KNr: 71	1. 4 Stühle Nr. 67
3	14.12.2020	Schulz - KNr: 33	1. 4 Tische Nr. 12, 2. 8 Stühle Nr. 67, 3. 1 Schrank Nr. 88

Anhand dieser Tabelle werden die ersten drei Normalformen erläutert. Dabei wird die Tabelle Schritt für Schritt in die dritte Normalform (3. NF) überführt.

LE 1 - Erste Normalform (1. NF)

Erste Normalform

Eine Tabelle liegt dann in der ersten Normalform vor, wenn alle Attribute der Tabelle nur einfache Werte aufweisen, d. h. wenn die Werte atomar vorliegen.

Die Beispieltabelle entspricht noch nicht der 1. NF, da sowohl in der Spalte „Kunde“ als auch in der Spalte „Bestellpositionen“ eine Aufzählung von Werten zu finden ist. Um die Tabelle in die 1. NF zu überführen, wird für jeden Wert, der in den Aufzählungen vorkommt, eine neue Spalte angelegt. Dazu wird die Spalte „Kunde“ in die Spalten „Name“ (Kundenname) und „KNr“ (KundenNr.) aufgeteilt. Aus der Spalte „Bestellpositionen“ entstehen die Spalten „Pos“ (Bestellposition), „Anzahl“, „Bezeichnung“ (Artikelbezeichnung) und „ANr“ (ArtikelNr.). Die Spalte „BestellNr“ ist in der neuen Tabelle nicht mehr eindeutig und wird um die Spalte „Pos“ erweitert. Somit ergibt sich der Primärschlüssel aus den Spalten „BestellNr“ und „Pos“, erkennbar an der Unterstreichung. Die Beispieltabelle sieht in der 1. NF wie folgt aus:

Beispiel in der ersten Normalform

Tabelle „Bestellungen“

<u>BestellNr</u>	<u>Pos</u>	Datum	KNr	Name	ANr	Bezeichnung	Anzahl
1	1	12.12.2020	71	Maier	12	Tisch	2
1	2	12.12.2020	71	Maier	88	Schrank	3
2	1	14.12.2020	71	Maier	67	Stuhl	4
3	1	14.12.2020	33	Schulz	12	Tisch	4
3	2	14.12.2020	33	Schulz	67	Stuhl	8
3	3	14.12.2020	33	Schulz	88	Schrank	1

Diese Tabelle enthält trotz erster Normalform noch viele Redundanzen. Redundanzen sind etwa in den Spalten „Datum“, „KNr“, „Name“, „ANr“ und „Bezeichnung“ erkennbar, die durch die nächsten Normalisierungsschritte beseitigt werden.

LE 2 - Zweite Normalform (2. NF)

Zweite Normalform

Eine Tabelle liegt dann in der zweiten Normalform vor, wenn sie der ersten Normalform genügt und alle Nichtprimärschlüsselattribute vom gesamten Primärschlüssel abhängig sind.

Die zweite Normalform wird angewendet, wenn eine Tabelle einen zusammengesetzten Primärschlüssel besitzt. Die Beispieltabelle liegt nicht in der 2. NF vor, da die Attribute „Datum“, „KNr“ und „Name“ nur vom Attribut „BestellNr“, also nicht vom gesamten Primärschlüssel abhängig sind. Auch das Datum einer Bestellung kann ermittelt werden, wenn man nur die „BestellNr“ kennt. Um einen bestimmten Artikel einer Bestellung zu ermitteln, sind beide Angaben des Primärschlüssels erforderlich, also „BestellNr“ und „Pos“. Die Tabelle wird in die 2. NF überführt, indem alle Attribute in eine neue Tabelle ausgelagert werden, die nur von einem Teil des Primärschlüssels abhängig sind. Auf diese Weise entstehen in unserem Beispiel zwei Tabellen, die durch eine Im-Beziehung über die „BestellNr“ miteinander verknüpft sind.

Beispiel in der zweiten Normalform

Tabelle „Bestellung“

BestellNr	Datum	KNr	Name
1	12.12.2020	71	Maier
2	14.12.2020	71	Maier
3	14.12.2020	33	Schulz

Tabelle „BestellPosition“

BestellNr	Pos	ANr	Bezeichnung	Anzahl
1	1	12	Tisch	2
1	2	88	Schrank	3
2	1	67	Stuhl	4
3	1	12	Tisch	4
3	2	67	Stuhl	8
3	3	88	Schrank	1

LE 3 - Dritte Normalform (3. NF)

Dritte Normalform

Eine Tabelle liegt dann in der dritten Normalform vor, wenn sie der zweiten Normalform genügt und kein Nichtschlüsselattribut transitiv abhängig ist.

Transitive Abhängigkeiten sind vermittelte oder berechnete Abhängigkeiten, die z. B. zwischen dem Nettopreis und dem Bruttopreis bei eindeutigem Mehrwertsteuersatz bestehen. Die beiden erzeugten Beispieltabellen befinden sich nicht in der 3. NF, da sie transitive Abhängigkeiten enthalten. In der Tabelle „Bestellung“ sind „KNr“ und „Name“ voneinander abhängig. In der Tabelle „BestellPosition“ besteht Abhängigkeit zwischen „ANr“ und „Bezeichnung“. Um die beiden Tabellen in die 3. NF zu überführen, werden die transitiv abhängigen Attribute jeweils in eine neue Tabelle ausgelagert, aber mit den alten Tabellen über eine Im-Beziehung verbunden. Damit entsprechen alle Tabellen der 3. NF und die Datenredundanz ist beseitigt.

Beispiel in der dritten Normalform

Tabelle „Bestellung“

BestellNr	Datum	KNr
1	12.12.2020	71
2	14.12.2020	71
3	14.12.2020	33

Tabelle „BestellPosition“

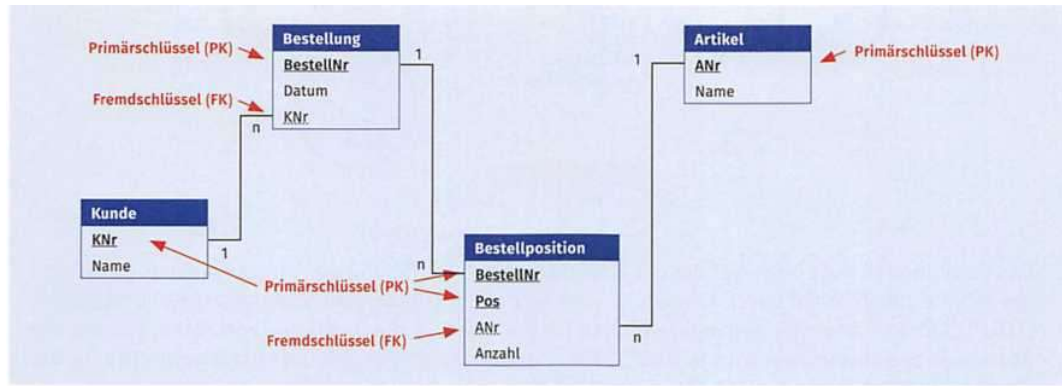
BestellNr	Pos	ANr	Anzahl
1	1	12	2
1	2	88	3
2	1	67	4
3	1	12	4
3	2	67	8
3	3	88	1

Tabelle „Kunde“

KNr	Name
71	Maier
33	Schulz

Tabelle „Artikel“

ANr	Bezeichnung
12	Tisch
67	Stuhl
88	Schrank



LE 4 - Weitere Normalformen und Nachteile der Normalisierung

Weitere Normalformen	
Normalform	Beschreibung
Boyce-Codd-Normalform (BCNF)	nur Abhängigkeiten vom Schlüssel zugelassen
Vierte Normalform (4. NF)	keine mehrwertigen Abhängigkeiten
Fünfte Normalform (5. NF)	nur triviale Verbundabhängigkeit

Auf Tabellen können noch weitere Normalformen angewandt werden, die in der Übersicht kurz zusammengefasst sind. Diese haben für die Praxis aber nicht die Bedeutung wie die ersten drei Normalformen. Deswegen werden sie hier auch nicht weiter vertieft. Außerdem entstehen mit jeder Normalisierungsstufe mehr Relationen, was auch zu einer Übernormalisierung führen kann. Mit einer zu starken Normalisierung können auch Nachteile verbunden sein.

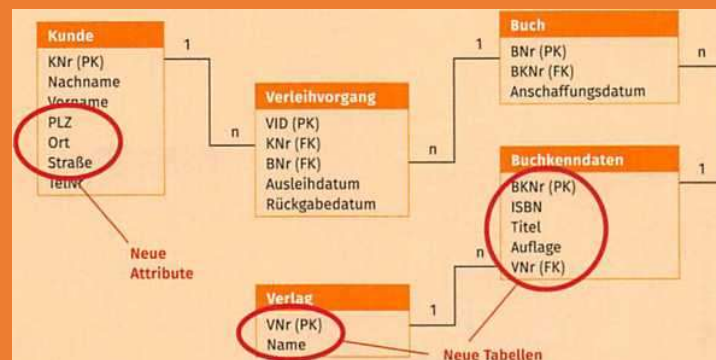
Nachteile einer zu starken Normalisierung

- Es entstehen viele kleine Tabellen, die die Leistung (z. B. das Antwortverhalten) der Datenbank negativ beeinflussen können.
- Aufgrund der vielen künstlichen Schlüssel und der erforderlichen zusätzlichen Verknüpfungen wird das System komplexer, was zu größerer Fehleranfälligkeit führen kann.
- Zusätzliche Schlüssel erfordern zusätzlichen Speicherplatz, stellen also auch eine Art von Redundanz dar.

Es ist daher immer ein sinnvoller und ausgewogener Kompromiss zwischen Redundanzfreiheit der Daten und der Performance des Systems anzustreben. Nichtjede, sondern nur die „unkontrollierte“ Redundanz sollte entfernt werden.

LE 5 - Weitere Beispielaufgabe

Normalisierung des Beispiels



Das Datenmodell liegt noch nicht in der 3. NF vor, weil in der Tabelle die Adresse nicht der 1. NF und die Tabelle „Buch“ nicht der 3. NF genügt. Hier sind die Attribute „VNr“, „Verlagsname“ und „ISBN“, „Titel“, „Auflage“ transitiv abhängig. Um das Datenmodell in die 3. NF zu überführen, werden die Adresse in die atomaren Attribute „PLZ“, „Ort“, „Straße“ aufgeteilt und die transitiven Attribute der Tabelle „Buch“ in zwei extra Tabellen ausgelagert.

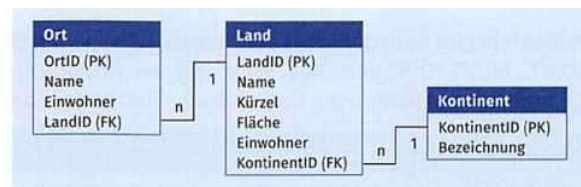
TAG 24 - Das Basiswissen über SQL erweitern und anwenden

Grundlage für diesen Abschnitt bilden die Inhalte über SQL aus dem Modul F „Software zur Verwaltung von Daten anpassen“. Die nachfolgende Tabelle gibt einen Überblick über die wichtigsten Anweisungen.



Basiswissen SQL - wichtige Anweisungen	
Anlegen und Löschen einer Datenbank	CREATE DATABASE ..., DROP DATABASE ...
Anlegen, Ändern und Löschen von Tabellen	CREATE TABLE ..., ALTER TABLE ... (ADD, MODIFY, DROP), DROP TABLE ...
Einfügen, Ändern und Löschen von Datensätzen	INSERT INTO ..., UPDATE ..., DELETE ...
Abfragen von Daten, Aggregatfunktionen, Vergleichs-, Rechen- und Logische Operatoren	SELECT ... FROM ... WHERE ..., LIKE ..., ORDER BY..., GROUP BY ..., MIN, MAX, SUM, COUNT, AVG, NULL

Im nachfolgenden Abschnitt wird vor allem das Wissen im Bereich von SQL-Abfragen vertieft und erweitert. Als Grundlage für alle Beispiele dient ein relationales Datenmodell der Beispieldatenbank „Ortsverwaltung“. Aber auch die Rechtevergabe wird hier behandelt.



LE 1 - Abfragen über mehrere Tabellen

In vielen Fällen werden Informationen nicht nur aus einer, sondern aus mehreren Tabellen benötigt. Um dieses in SQL-Anweisungen umzusetzen, gibt es zwei Wege. In der klassischen Variante werden alle benötigten Tabellen in der FROM-Klausel angegeben und in der WHERE-Klausel die Primärschlüssel mit den entsprechenden Fremdschlüsseln gleichgesetzt. In der modernen Variante geschieht dieses mithilfe des Schlüsselworts „INNER JOIN“ in der FROM-Klausel. Im Beispiel einer SQL-Abfrage über zwei Tabellen sollen alle Länder ausgegeben werden, die in Europa liegen. Die Abfrage in der klassischen Variante würde wie folgt aussehen.

Beispiel: Abfragen über mehrere Tabellen klassische Variante

```
</> SELECT L.Name, K.Bezeichnung FROM Land L, Kontinent K WHERE L.KontinentID = K.KontinentID AND K.Bezeichnung = 'Europa';
```

Die gleiche Ergebnismenge erhält man auch mithilfe von INNER JOIN.

Beispiel: Abfragen über mehrere Tabellen mit INNER JOIN

```
</> SELECT L.Name, K.Bezeichnung FROM Land L INNER JOIN Kontinent K ON L.KontinentID = K.KontinentID WHERE K.Bezeichnung = 'Europa';
```

Im nächsten Beispiel wird noch einmal eine SQL-Abfrage über drei Tabellen vorgenommen. Das Verfahren bleibt das Gleiche, nur dass sich in diesem Fall die Anzahl der Tabellen und der Verknüpfungsbedingungen erhöht. Für die Abfrage sollen alle Orte ausgegeben werden, die in Europa liegen und mit einem „B“ beginnen, z. B. Berlin. Die Abfrage in der klassischen Variante würde wie folgt aussehen:

Beispiel: Abfragen über mehrere Tabellen klassische Variante

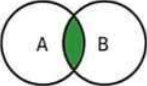
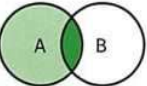
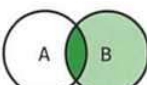
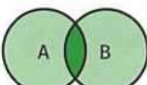
```
</> SELECT O.Name, K.Bezeichnung FROM Ort O, Land L, Kontinent K
WHERE O.LandID = L.LandID AND L.KontinentID = K.KontinentID AND
K.Bezeichnung = 'Europa' AND O.Name LIKE 'B%';
```

Die gleiche Ergebnismenge erhält man auch mithilfe von INNER JOIN.

Beispiel: Abfragen über mehrere Tabellen mit INNER JOIN

```
</> SELECT O.Name, K.Bezeichnung
FROM Ort O INNER JOIN Land L ON O.LandID = L.LandID
INNER JOIN Kontinent K ON L.KontinentID = K.KontinentID
WHERE K.Bezeichnung = 'Europa' AND O.Name LIKE 'B%';
```

Außerdem gibt es in der modernen Variante neben „INNER JOIN“ auch noch die Schlüsselwörter „LEFT JOIN“, „RIGHT JOIN“ und „FULL JOIN“, die wie INNER JOIN verwendet werden, aber eine unterschiedliche Ergebnismenge liefern. Die Unterschiede sind in der nachfolgenden Tabelle nochmal aufgeführt.

Schlüsselwort	Beschreibung	Beispiel
INNER JOIN	Liefert die Werte der linken Tabelle und die Werte der rechten Tabelle, welche miteinander verknüpft sind. 	<pre>SELECT ... FROM Land L INNER JOIN Kontinent K ON L.KontinentID = K.KontientID</pre>
LEFT JOIN	Liefert alle Werte der linken Tabelle und alle Werte der rechten Tabelle, die eine Verknüpfung mit der linken Tabelle besitzen. 	<pre>SELECT ... FROM Land L LEFT JOIN Kontinent K ON L.KontinentID = K.KontientID</pre>
RIGHT JOIN	Liefert alle Werte der rechten Tabelle und alle Werte der linken Tabelle, die eine Verknüpfung mit der rechten Tabelle besitzen. 	<pre>SELECT ... FROM Land L RIGHT JOIN Kontinent K ON L.KontinentID = K.KontientID</pre>
FULL JOIN	Liefert alle Werte der rechten und der linken Tabelle, unabhängig davon, ob eine Verknüpfung besteht oder nicht. 	<pre>SELECT ... FROM Land L FULL JOIN Kontinent K ON L.KontinentID = K.KontientID</pre>

„LEFT JOIN“, „RIGHT JOIN“ und „FULL JOIN“ sind für bestimmte Abfragen sehr nützlich. Wenn es z. B. eine Tabelle „Mitarbeiter“ und eine Tabelle „Projektarbeitszeit“ in der die Arbeitsstunden am Projekt von den einzelnen Mitarbeitern erfasst werden, gibt, dann liefert eine Abfrage über die geleisteten Stunden aller Mitarbeiter mithilfe von „INNER JOIN“ unter Umständen ein unvoll-

ständiges Ergebnis. Dieses ist immer dann der Fall, wenn einige Mitarbeiter noch keine Stunden eingetragen haben. Diese fallen dabei durch das Raster, weil noch keine Verknüpfung zwischen den beiden Tabellen für diese Mitarbeiter existiert. In diesem Fall bietet sich der Einsatz von „LEFT JOIN“ an. Hier werden alle Mitarbeiter aufgelistet, auch wenn noch keine Einträge vorhanden sind.

LE 2 - Kombinieren von Abfragemengen

Mit dem UNION-Befehl kann man die Ergebnismengen von zwei oder mehr Abfragen kombinieren. Damit ähnelt UNION in gewisser Hinsicht dem Schlüsselwort JOIN, da beide für aufeinander bezogene Daten in unterschiedlichen Tabellen verwendet werden. Bei UNION gilt jedoch die Einschränkung, dass alle entsprechenden Spalten vom gleichen Datentyp sein müssen. Außerdem werden doppelte Werte, ähnlich wie bei DISTINCT, ignoriert. Als Beispiel soll eine Abfrage alle Namen anzeigen, welche für Orte und für Länder verwendet werden. Allerdings würde der Name „Luxemburg“ nur einmal auftauchen, weil doppelte Werte ignoriert werden. Soll der Name „Luxemburg“ sowohl als Name des Landes als auch als Stadtname in der Ergebnismenge erscheinen, dann muss der Befehl „UNION ALL“ verwendet werden.

Kombinieren von Abfragemengen mit UNION	
Syntax	<code>SELECT Spaltenname FROM TabellennamenA UNION SELECT Spaltenname FROM TabellennamenB;</code>
Erläuterung	Spaltenname = Name der Spalte in der Tabelle A oder der Tabelle B TabellennamenA = Name der ersten Tabelle, aus der die Daten abgefragt werden TabellennamenB = Name der zweiten Tabelle, aus der die Daten abgefragt werden
Beispiel	<code>SELECT O.Name FROM Ort O UNION SELECT L.Name FROM Land L;</code>

LE 3 - Unterabfragen

Unterabfragen (Sub-Selects, Sub-Queries) werden dazu verwendet, um SQL-Anweisungen in andere SQL-Anweisungen einzubinden. Dadurch können Ergebnisse der Unterabfrage in der Haupt-SQL-Anweisung verwendet werden. Dabei werden Unterabfragen unterschieden in solche, die nur einen Wert zurückgeben, und solche, die mehrere Werte zurückgeben. Bei Unterabfragen, welche nur einen Wert zurückgeben, wird hauptsächlich mit Vergleichsoperatoren wie =, >, >=, < oder <= gearbeitet. Man kann diese Unterabfragen auch gut für Aggregatfunktionen wie MAX oder AVG verwenden. Als Beispiel sollen alle Orte angezeigt werden, deren Einwohnerzahl über der durchschnittlichen Einwohnerzahl aller Orte liegt.

```
</> SELECT O.Name, O.Einwohner FROM Ort O
WHERE O.Einwohner > (SELECT AVG(O.Einwohner) FROM Ort O)
```

Unterabfragen können in allen Arten von SQL-Anweisungen verwendet werden. Ein Beispiel dafür ist, wenn alle Orte des Landes „Malta“ gelöscht werden sollen.

Wenn Unterabfragen mehr als einen Wert zurückgeben, dann ist der Einsatz von Mengenoperatoren (ALL, ANY, IN, EXISTS) notwendig. Einen Überblick über diese Mengenoperatoren liefert

```
</> DELETE FROM Ort O
WHERE O.LandID = (SELECT L.LandID FROM Land L WHERE L.Name = 'Malta')
```

folgende Tabelle.

Operator	Beschreibung
IN	Prüft, ob ein bestimmter Wert in einer Menge von Werten der Unterabfrage enthalten ist.
ALL	Prüft, ob eine Bedingung für alle Ergebnisse der Unterabfrage erfüllt ist.
ANY	Prüft, ob eine Bedingung für ein beliebiges Ergebnis der Unterabfrage erfüllt ist.
EXISTS	Prüft, ob die Unterabfrage ein Ergebnis hat. Gibt den Wert „wahr“ zurück, wenn die Unterabfrage einen oder mehrere Datensätze liefert.

Für das folgende Beispiel einer Abfrage über zwei Tabellen ist zunächst keine Unterabfrage nötig. Es sollen Orte ermittelt werden, die den gleichen Namen haben wie das dazugehörige Land.

```
</> SELECT O.Name AS Ortsname, L.Name AS Landesname FROM Ort O INNER JOIN Land L ON O.LandID = L.LandID WHERE O.Name = L.Name;
```

Sollen aber alle Orte ermittelt werden, deren Name mit irgendeinem beliebigen Land übereinstimmt, dann kann dieses nur mithilfe einer Unterabfrage und dem IN-Operator geschehen. Das folgende Beispiel zeigt eine entsprechende Unterabfrage mit IN-Operator.

```
</> SELECT O.Name AS Ortsname, L.Name AS Landesname FROM Ort O INNER JOIN Land L ON O.LandID = L.LandID WHERE O.Name IN (SELECT L.Name FROM Land L)
```

Unterabfragen können an verschiedenen Stellen in einer SQL-Anweisung, z. B. in der SELECT- oder der WHERE-Klausel, eingesetzt werden. Außerdem ist es möglich, dass Unterabfragen wiederum Unterabfragen enthalten. Wie sinnvoll dieses Vorgehen ist, hängt ganz von der Aufgabenstellung ab. Zusammenfassend gelten für Unterabfragen aber immer folgende Regeln:

Regeln für Unterabfragen

1. Eine Unterabfrage wird in runde Klammern gesetzt.
2. Für einen einzelnen Datensatz als Rückgabewert können Vergleichsoperatoren verwendet werden.
3. Für mehrere Datensätzen als Rückgabewert müssen Mengenoperatoren verwendet werden.
4. ORDER BY und UNION sind in Unterabfragen unzulässig.

Durch Abfragen über mehrere Tabellen und Unterfragen können mitunter sehr komplexe SQL-Anweisungen entstehen. Dabei sollte immer beachtet werden, dass komplexe SQL-Anweisungen immer schwer zu warten sind. In vielen Fällen gibt es auch elegante Alternativen dazu.

LE 4 - Rechtevergabe

Besonders in einer Mehrbenutzerumgebung ist es notwendig, den Zugang zur Datenbank und die Rechte an den einzelnen Datenbankobjekten zu verwalten. Rechte können z. B. zu den Befehlen INSERT, SELECT, ALTER, DELETE usw. vergeben werden. Nachfolgend werden die wichtigsten Befehle vorgestellt.

Anlegen eines Benutzers	
Syntax	CREATE USER Benutzername IDENTIFIED BY Passwort;
Erläuterung	Benutzername = Zugangsname des Benutzers Passwort = Zugangscode des Benutzers
Beispiel	CREATE USER Schulz IDENTIFIED BY '123xyz456';

Löschen eines Benutzers	
Syntax	<code>DROP USER Benutzername;</code>
Erläuterung	Benutzername = Name des Benutzers, der gelöscht werden soll.
Beispiel	<code>DROP USER Schulz;</code>

Erteilen von Rechten	
Syntax	<code>GRANT Rechtesteliste ON Objektname TO Benutzername [WITH GRANT OPTION];</code>
Erläuterung	Rechtesteliste = Aufzählung aller Rechte, die der Benutzer erhalten soll Objektname = Name des Datenbankobjektes Benutzername = Name des Benutzers WITH GRANT OPTION = Damit darf der Benutzer seine Rechte weitergeben.
Beispiel	<code>GRANT INSERT, UPDATE ON Ortsverwaltung.Ort TO Schulz;</code>

Entziehen von Rechten	
Syntax	<code>REVOKE Rechtesteliste ON Objektname FROM Benutzername;</code>
Erläuterung	Rechtesteliste = Aufzählung aller Rechte, die dem Benutzer entzogen werden sollen Objektname = Name des Datenbankobjektes Benutzername = Name des Benutzers
Beispiel	<code>REVOKE INSERT, UPDATE ON Ortsverwaltung.Ort FROM Schulz;</code>

TAG 25 - Eine MySQL-Datenbank mit Java ansprechen

MySQL ist ein relationales Datenbankmanagementsystem und zählt zu den verbreitetsten Datenbanksystemen der Welt. Es gibt eine Open-Source und eine kommerzielle Variante, welche für verschiedene Betriebssysteme zur Verfügung stehen. Da MySQL in der Lage ist, große Datenmengen schnell und performant zu verarbeiten, wird es in vielen Bereichen eingesetzt. So bildet dieses Datenbanksystem beispielsweise in vielen Content-Management-Systemen (WordPress, TYPO3 usw.) die technische Grundlage für die Speicherung der Daten. Es wurde im Jahr 1994 von dem schwedischen Unternehmen MySQLAB entwickelt und wechselte seitdem einige Male den Besitzer, bis MySQL 2010 von der Firma Oracle übernommen wurde, welches es bis heute kontinuierlich weiterentwickelt. MySQL ist auf eine platzsparende Datenspeicherung ausgelegt, wird mithilfe von SQL angesprochen und arbeitet nach dem Client-Server-Prinzip bestehend aus einem MySQL-Server und einem oder mehreren MySQL-Clients. Allerdings besitzt MySQL keine eigene grafische Benutzerschnittstelle, sondern es müssen dafür die Lösungen von externen Anbietern genutzt werden.



Die aktuelle Version von MySQL kann unter <https://dev.mysql.com/downloads/> [15.06.2021] heruntergeladen und danach installiert werden.

Nach der Installation von MySQL kann die Datenbank genutzt werden. Um eine Verbindung aus einem Java-Programm heraus herzustellen, muss MySQL auf einem erreichbaren Rechner laufen. Liegt die Datenbank auf einem anderen Rechner oder sogar in der Cloud, dann braucht der Benutzer die entsprechenden Rechte, um von außerhalb auf die Daten zuzugreifen. Um eine MySQL-Datenbank in einem Java-Programm anzusprechen, müssen zunächst entsprechende Treiber geladen und eine Verbindung aufgebaut werden. Die entsprechenden Befehle dafür lauten wie folgt:

```
</> // Laden der Treiberklasse
Class.forName("com.mysql.jdbc.Driver");

// Erzeugen eines Verbindungsobjektes
Connection con = DriverManager.getConnection("jdbc:mysql://Host:Port/Datenbank",
                                             "Benutzer", "Passwort");
```

Nachfolgend ist ein Programmbeispiel aufgeführt, wie mithilfe eines Java-Konsolenprogramms eine Verbindung zu der MySQL-Beispieldatenbank „Ortverwaltung“ aus dem vorherigen Abschnitt hergestellt wird und danach SQL-Anweisungen auf dieser ausgeführt werden.

Beispiel für das Herstellen einer Verbindung zu einer MySQL-Datenbank mithilfe von Java

```

</> // -----
//   Klasse MySQLConnection
// -----
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class MySQLConnection {
    private Connection con = null;
    private Statement sqlStatement = null;
    private String dbHost = "localhost"; // Hostname
    private String dbPort = "3306"; // Port -- Standard: 3306
    private String dbName = "Ortverwaltung"; // Datenbankname
    private String dbUser = "user"; // Benutzername
    private String dbPass = "password"; // Datenbankpasswort

    public void create()
    {
        try {
            // Datenbanktreiber für JDBC Schnittstellen laden.
            Class.forName("com.mysql.jdbc.Driver");
            // Verbindung zur JDBC-Datenbank herstellen.
            con = DriverManager.getConnection("jdbc:mysql://" + dbHost +
                ":" + dbPort + "/" +
                dbName + "?" + "user=" + dbUser + "&" +
                "password=" + dbPass);

            sqlStatement = con.createStatement();
            System.out.println("Verbindung erfolgreich hergestellt.");
        } catch (ClassNotFoundException e) {
            System.out.println("Verbindung nicht möglich");
        } catch (SQLException e) {
            System.out.println("Verbindung nicht möglich");
            System.out.println("SQLException: " + e.getMessage());
            System.out.println("SQLState: " + e.getSQLState());
            System.out.println("VendorError: " + e.getErrorCode());
        }
    }

    public void close() {
        try {
            if (sqlStatement != null) { sqlStatement.close(); }
            if (con != null) {
                con.close();
                System.out.println("Datenbank erfolgreich geschlossen");
            }
        } catch (SQLException e) {
            System.out.println("SQLException: " + e.getMessage());
            System.out.println("SQLState: " + e.getSQLState());
            System.out.println("VendorError: " + e.getErrorCode());
        }
    }
}

```

Notwendige Klassen importieren

Klassenvariablen anlegen und initialisieren. Die Werte müssen an die konkreten Bedingungen angepasst werden.

Datenbankverbindung herstellen

Exceptions abfangen und auswerten

Datenbankverbindung schließen

```

public void updateDB(String sql) {
    try {
        sqlStatement.executeUpdate(sql);
    } catch (SQLException e) {
        System.out.println ("SQLException: " + e.getMessage());
        System.out.println ("SQLState: " + e.getSQLState());
        System.out.println ("VendorError: " + e.getErrorCode());
    }
}

public ResultSet getData(String sql) {
    try {
        return sqlStatement.executeQuery(sql);
    } catch (SQLException e) {
        System.out.println ("SQLException: " + e.getMessage());
        System.out.println ("SQLState: " + e.getSQLState());
        System.out.println ("VendorError: " + e.getErrorCode());
        return null;
    }
}

// -----
//   Hauptprogramm
// -----
public class Program {
    public static void main(String[] args) {
        MySqlConnection mySqlConnection = null;
        try {
            mySqlConnection = new MySqlConnection();
            mySqlConnection.create();

            String sqlStatement = "CREATE TABLE Kontinent " +
                                   "(KontinentID INTEGER not NULL, " +
                                   " Bezeichnung VARCHAR(100), " +
                                   " PRIMARY KEY ( KontinentID ))";

            mySqlConnection.updateDB(sqlStatement);
            System.out.println ("SQL-Anweisung erfolgreich ausgeführt");
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (mySqlConnection != null) { mySqlConnection.close(); }
        }
    }
}

```

Datenbank mit der übergebenen SQL-Anweisung aktualisieren; es können u. a. folgende Anweisungen ausgeführt werden: INSERT INTO, UPDATE, DELETE, CREATE TABLE, ALTER TABLE, DROP TABLE.

Es werden Daten aus der Datenbank mithilfe einer SELECT-Anweisung abgefragt. Die Ergebnismenge wird als ResultSet an das Programm zurückgegeben und kann dann weiterverarbeitet werden.

Im Hauptprogramm wird beispielhaft eine Datenbankverbindung aufgebaut und die Tabelle „Kontinent“ aus dem Beispieldatenmodell (S. 275) erzeugt.

Werden Daten aus der Datenbank abgefragt, dann wird die Ergebnismenge in einem ResultSet-Objekt zurückgegeben. Hier können die Daten dann mithilfe einer Schleife verarbeitet werden.

Beispiel für das Auswerten eines ResultSets

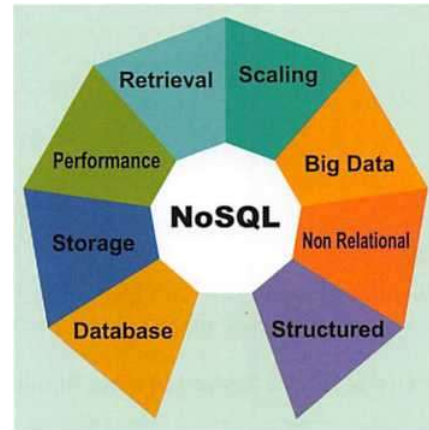
```

</> ResultSet rs = sqlStatement.executeQuery("SELECT O.Name, O.Einwohner, L.Name " +
        "FROM Ort O INNER JOIN Land L ON O.LandID = L.LandID");
while (rs.next()) {
    String ortsName = rs.getString("Name");
    int anzahlEinwohner = rs.getInt("Einwohner");
    String landName = rs.getString("Name");
    // Ausgabe der Werte
    System.out.format("%s, %s, %s\n", ortsName, anzahlEinwohner, landName);
}

```

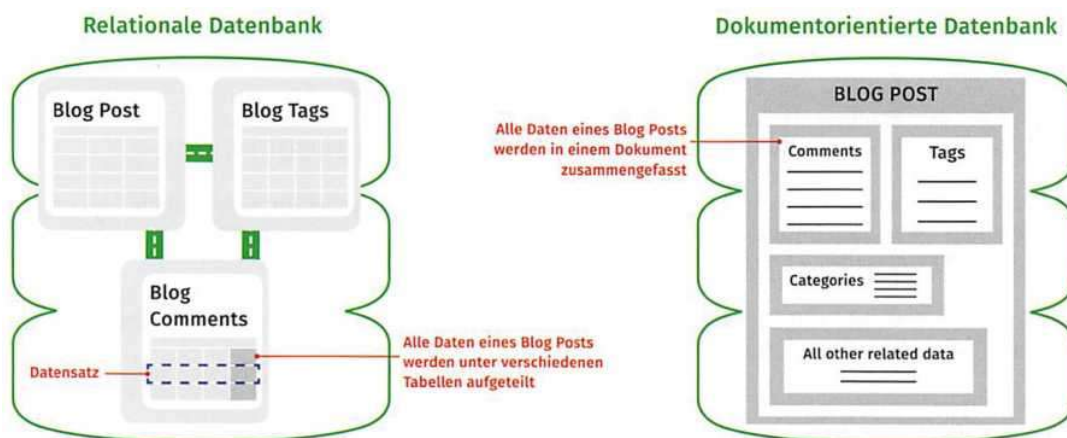
TAG 26 - NoSQL-Datenbanken und deren Datenmodelle unterscheiden

NoSQL-Datenbanken (Not only SQL) sind keineswegs als Ersatz für die traditionellen, relationalen Datenbanken gedacht, sondern sollen diese in Bereichen ergänzen, in denen die klassischen relationalen Datenbanken an ihre Grenzen stoßen. Dazu wurden neue Datenmodelle entwickelt, welche zur Speicherung und Verarbeitung unstrukturierter und teilweise auch unbekannter Daten, wie sie z. B. im Big-Data-Bereich anfallen, möglich wird. Anders als relationale Datenbanken nutzen NoSQL-Datenbanken keine Tabellen mit Zeilen und Spalten zur Datenspeicherung. Stattdessen organisieren sie große Datenmengen mithilfe flexibler Techniken. Obwohl es keine einheitlichen Regelungen gibt, lassen sich die verschiedenen NoSQL-Ansätze in vier Hauptkategorien unterteilen.



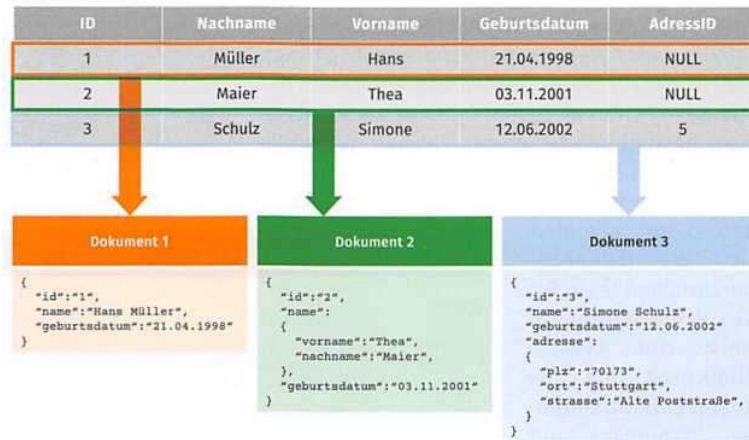
LE 1 - Dokumentenorientierte Datenbank

In dokumentenorientierten Datenbanken werden die Daten in Form von Dokumenten abgespeichert. Dabei stellt jeder einzelne Datensatz ein Dokument dar und dieses kann durch einen eindeutigen Identifikator angesprochen werden. Die Daten in den Dokumenten müssen nicht strukturiert sein und können sich von Dokument zu Dokument erheblich unterscheiden. Ihnen werden verschiedene Attribute zugeordnet, mit deren Hilfe die Inhalte des Dokuments durchsucht werden können. Soll z. B. bei einem Dokument eine zusätzliche Information hinzugefügt werden, dann kann dieses ohne großen Aufwand geschehen, weil die Dokumente keinen vorgefertigten Dateityp haben.



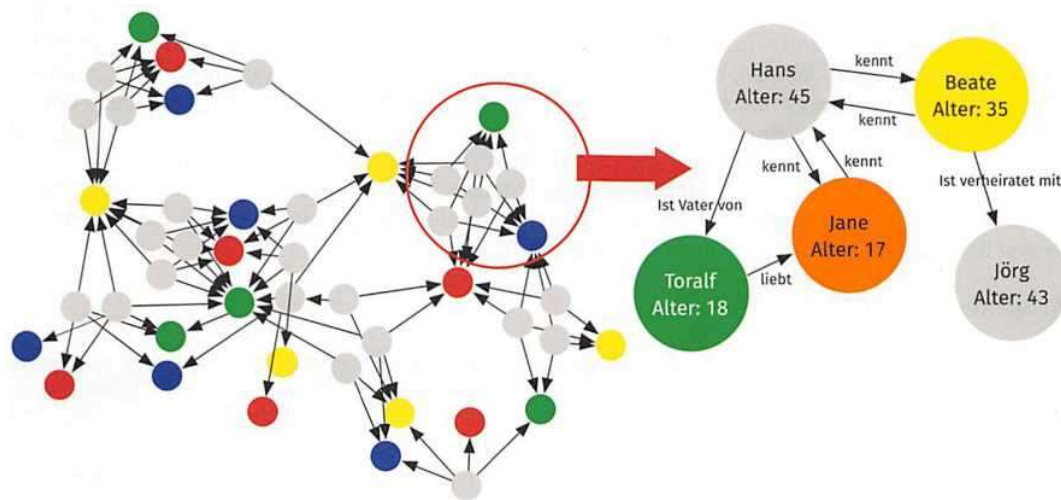
In relationalen Datenbanken wäre dies undenkbar, da jeder Datensatz in einer Tabelle einem festen Muster unterliegt. Dokumentenorientierte Datenbanken kommen immer dann zum Einsatz, wenn für die zu speichernden Daten kein gemeinsamer Typ festgelegt werden kann und die gespeicherten Informationen sich in vielen Datensätzen unterscheiden. Somit eignen sich dokumentenorientierte Datenbanken besonders gut für Content-Management-Systeme und

Blogs. Als Datenformat wird heute vor allem JSON verwendet, dass den schnellen Datenaustausch zwischen Anwendungen ermöglicht. Um Referenzen darzustellen, kann beispielsweise ein neues Dokument in ein vorhandenes Dokument als ein neuer Unterpunkt eingehängt werden. Beispiele für Dokumentendatenbanken sind Terastore, Amazon Document-DB, Google Cloud Firestone oder Apache CouchDB.



LE 2 - Graphdatenbank

Graphdatenbanken speichern Daten mithilfe von Knoten (Eckpunkte), Kanten (Beziehungen) und Eigenschaften. Eine Beziehung stellt immer eine Verbindung zwischen zwei Knoten dar und bildet ab, wie die Entitäten genau in Beziehung stehen, z. B. durch Aktionen, Besitz, Abstammung, Kaufoptionen, persönliche Verbindungen, Familienbindungen usw. Dies macht sie nützlich für die Modellierung vieler Aspekte der realen Welt.



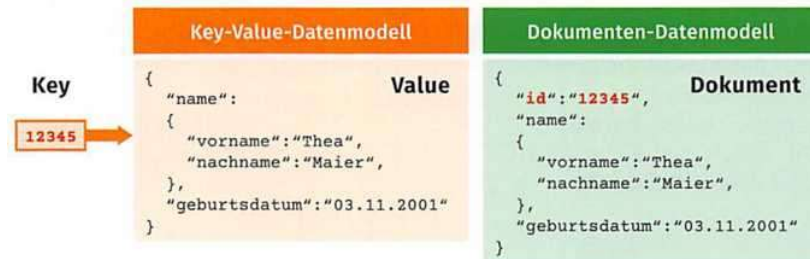
Beziehungen besitzen einen Start- und einen Endknoten, einen Typ und eine Richtung. Auf diese Weise wird das Beziehungsgeflecht der Daten abgebildet. Außerdem können die Knoten und Kanten noch zusätzliche Attribute besitzen. Graphdatenbanken werden vor allem in Bereichen mit stark vernetzten Informationen wie z.B. im Social-Media-Bereich oder dem Bereich der Betrugserkennung verwendet. Um z. B. die Beziehungen zwischen Followern auf Twitter oder Instagram mithilfe von relationalen Datenbanken darzustellen, müssen mehrere Tabellen mit mehreren Fremdschlüsseln erstellt werden und anschließend die Daten mit verschachtelten SQL-Abfragen und komplexen JOINS abgerufen werden. Dies wird sehr schnell unübersichtlich und die Leistung nimmt mit zunehmender Datenmenge ab. Hier erzielen Graphdatenbanken eine deutlich bessere Performance als relationale Datenbanken. Beispiele für Graphdatenbanken sind Neptune, OrientDB oder Neo4j.

LE 3 - Key-Value-Datenbank

Key-Value-Datenbanken speichern die Daten in Form von Schlüssel-Werte-Paaren in einer Hash- Tabelle. Jeder Wert (Value) kann sowohl Strings, Listen oder Hashes enthalten und wird einem bestimmten Schlüssel zugeordnet. Dieser Schlüssel (Key) besteht aus beliebigen

Zeichen, welche in Form einer Zeichenkette abgebildet wird. Der Schlüssel ist immer eindeutig und bildet zugleich den Index, mit dem sich die Datenbank durchsuchen lässt. Die Key-Value-Datenbank weist eine gewisse Ähnlichkeit mit der dokumentenorientierten Datenbank auf, welche jedoch den Key mit in dem Dokumentspeichert, während in der Key-Value-Datenbank der Key nicht im Datensatz selbst vorhanden ist, sondern lediglich zu diesem führt. Aus diesem einfachen Schema der Datenspeicherung erwachsen die großen Vorteile von Key-Value-Datenbanken. Diese bestehen in dem schnellen und einfachen Zugriff auf die einzelnen Datensätze, was zu einer sehr schnellen Datenverarbeitung und einer hohen Skalierbarkeit des Systems führt. Bei riesigen Datenmengen kann die Last ganz einfach auf mehrere Server verteilt werden. Dieses nennt man auch horizontale Skalierung. Diese Vorteile werden aber auch durch einige Nachteile erkauft. Weil der Zugriff einer Key-Value-Datenbank lediglich über den Schlüssel erfolgt, sind die Abfragemöglichkeiten der Daten teilweise sehr eingeschränkt. Um eine schnelle Schreib- und Lesegeschwindigkeit gewährleisten zu können, wird i.d.R. auf Indizes und komplexe Suchalgorithmen verzichtet. Für umfangreiche Abfragen muss auf eine API zurückgegriffen werden. Wenn mithilfe einer Key-Value Datenbank viele gegenseitige Beziehungen abgebildet werden sollen, dann wird es schnell unübersichtlich und komplex. In diesem Fall sollte man eine andere Datenbank benutzen. Spielen die Komplexität der Beziehungen und die Möglichkeit von komplexen Datenabfragen keine Rolle, dann sind Key-Value-Datenbanklösungen in puncto Geschwindigkeit und Skalierbarkeit unschlagbar. Beispiele für Datenbanken in diesem Bereich sind Amazon Dynamo oder Google BigTable.

Key	Value
12345	12.45,345.66, 445.99
12246	{vorname: "Uwe", nachname: "Meier"}
33444	Zitat: "Das hat folgende Bedeutung ..."
50078	"Anzahl Mitglieder: 51"



LE 4 - Spaltenorientierte Datenbank

Spaltenorientierte Datenbanken speichern die Daten nicht in Zeilen wie in relationalen Datenbanken, sondern in Spalten. Ansonsten ist der Aufbau ähnlich, nur dass die Daten anders im Speicher abgelegt werden. Relationale Datenbanken enthalten oftmals sehr viele Tabellen, welche wiederum aus einigen Spalten bestehen. Bei Abfragen wird aber oftmals nur ein Bruchteil der Spalten benötigt und der Aufwand, die Werte der entsprechenden Spalte aus den Datensätzen auszulesen, ist relativ hoch. Dadurch, dass bei spaltenorientierten Datenbanken alle Werte einer Spalte hintereinander gespeichert sind, ist dieser Vorgang sehr schnell.

ID	Nachname	Vorname	Geburtsdatum	Gehalt
1	Müller	Hans	21.04.1998	2700
2	Maier	Thea	03.11.2001	2220
3	Schulz	Simone	12.06.2002	1970

Zeilenorientierte Speicherung

1, Müller, Hans, 21.04.1998, 2700; 2, Maier, Thea, 03.11.2001, 2220; 3, Schulz, Simone, 12.06.2002, 1970;

Spaltenorientierte Speicherung

1, 2, 3; Müller, Maier, Schulz; Hans, Thea, Simone; 21.04.1998; 03.11.2001, 12.06.2002; 2700, 2220, 1970;

Der Zusammenhang der Attributwerte ist durch deren Position in der jeweiligen Spalte gegeben. Der Vorteil einer spaltenorientierten Datenbank kommt also dann besonders zum Tragen, wenn eher die Daten einer Spalte anstatt einer Zeile ausgelesen werden sollen. Ähnliches gilt auch für das Aktualisieren von vielen Daten einer Spalte. Die spaltenorientierte Datenhaltung hat ihre Nachteile allerdings beim Ergänzen oder Lesen von Zeilen. Typischerweise werden spaltenorientierte Datenbanken für Data-Warehouse oder für Data-Mining und Analyseprogramme (OLAP) genutzt. Beispiele für spaltenorientierte Datenbanken sind Apache Cassandra oder Sybase IQ.

TAG 27 - Die NoSQL-Datenbank „MongoDB“ mit Python ansprechen

MongoDB ist ein universelles, dokument-basiertes NoSQL-Datenbankmanagementsystem für die moderne Anwendungsentwicklung und die Cloud. Es gehört zu den am meisten genutzten NoSQL-Datenbanken und ist sehr weit verbreitet. Das Format der Dokumente, welche in MongoDB verwaltet werden, ist dem JSON-Format sehr ähnlich. MongoDB ist Open-Source und für Windows, MacOS und Linux verfügbar. Bevor in Python mit MongoDB gearbeitet werden kann, muss das Paket „PyMongo“ installiert werden.



Die aktuelle Version von MongoDB kann unter www.mongodb.com/try/download/community [15.06.2021] heruntergeladen und danach installiert werden. Mitfolgender Konsolenanweisung kann das Paket PyMongo installiert werden: `python -m pip install pymongo`.

Nachdem alles installiert ist, kann eine Datenbank angelegt und mit Daten beschrieben werden. Das nachfolgende Beispiel zeigt exemplarisch diesen Prozess. Dazu ist folgendes Beispiel für einen Einstieg bewusst sehr einfach gehalten.

Beispiel für das Schreiben und Lesen von Dokumenten in einer MongoDB

```

</> from pymongo import MongoClient

database = MongoClient('mongodb://localhost:27017/')['artikelverwaltung_db']
collection = database['artikel']

schrank_daten = {
    'artikel_nr': '002348',
    'name': 'Holzschrank HS900',
    'hersteller': 'Möbel GmbH',
    'preis': '588.00 Euro'
}
collection.insert_one(schrank_daten)

tisch_daten = {
    'artikel_nr': '1234',
    'name': 'Holztisch HT73',
    'hersteller': 'Möbel GmbH',
    'preis': '213.45 Euro'
}

stuhl_daten = {
    'artikel_nr': '5401',
    'name': 'Holzstuhl HST08',
    'hersteller': 'Möbel GmbH',
    'preis': '43.71 Euro'
}
collection.insert_many([tisch_daten, stuhl_daten])

daten = collection.find_one({"artikel_nr": "1234"})
print("Daten des Artikels mit der Nr. 1234")
print(daten)

print("Alle Artikel des Herstellers -Möbel GmbH-")
for daten in collection.find({"hersteller": "Möbel GmbH"}):
    print(daten)

```

Notwendige Module importieren

Eine Datenbank „artikelverwaltung_db“ und dort eine Collection „artikel“ erzeugen. Eine Collection ist eine Gruppe von Dokumenten, welche ungefähr einer Tabelle in einem relationalen Modell entspricht.

Ein einzelnes Dokument erzeugen und in die Datenbank speichern. Dazu muss die Funktion „insert_one(...)“ der entsprechenden Collection verwendet werden.

Mehrere Dokumente erzeugen und in die Datenbank speichern. Dazu muss die Funktion „insert_many(...)“ der entsprechenden Collection verwendet werden.

Die Funktion „find_one(...)“ gibt genau ein Dokument zurück. Die Daten werden im Anschluss ausgegeben.

Die Funktion „find(...)“ gibt mehrere Dokumente zurück. Die Daten werden im Anschluss mithilfe einer Schleife angezeigt.

TAG 28 - Cloud-basierte Datenbanklösungen unterscheiden

Die Datenbanklandschaft ist nicht mehr nur auf die traditionellen Datenbankmanagementsysteme beschränkt. Cloud-Datenbanken gehören zu dem am schnellsten wachsenden Bereich des Cloud-Marktes und werden i.d.R. als Database as a Service (DBaaS) angeboten. Die Angebote reichen von universellen Services, welche einen breiten Anwendungsbereich abdecken, bis hin zu Speziallösungen, welche sich auf Nischenanforderungen konzentrieren. Es werden sowohl relationale als auch NoSQL-Datenbanken zur Nutzung angeboten.



DBaaS-Modell (Database as a Service)

Beim DBaaS-Modell schließt ein Unternehmen einen Vertrag mit einem Cloud-Services-Anbieter über die Nutzung einer Datenbank in der Cloud ab. Die Datenbank wird beim Service-Provider ausgeführt und dieser kümmert sich je nach Modell um den Betrieb, die Wartung und die Administration der Datenbank. Außerdem umfasst der Vertrag i.d.R. auch andere wichtige Bereiche des Datenbankmanagements wie Sicherung, Skalierung, Hochverfügbarkeit, Sicherheit, Patching und Integritätsüberwachung.

Cloudbasierte Datenbanklösungen bieten viele der Vorteile, welche sich mit denen von anderen cloudbasierten Services decken.

Vorteile von cloudbasierten Datenbanklösungen	
Flexibilität	Cloud-Datenbanken können sehr schnell in und außer Betrieb genommen werden. Auf diese Art und Weise kann sehr flexibel auf veränderte Rahmenbedingungen im Geschäftsumfeld reagiert werden.
Schnelle Markteinführung	Mithilfe von Cloud-Datenbanken ist es sehr einfach und schnell möglich, neue Geschäftsideen umzusetzen, z. B. im Bereich des E-Commerce, da kein Datenbankserver angeschafft und installiert werden muss. Der Datenbankzugriff kann innerhalb weniger Minuten erfolgen.
Geringe Risiken	Durch die Möglichkeiten, welche die DBaaS-Modelle bieten, werden die Risiken des Unternehmens erheblich reduziert. So können z.B. Umsatzverluste aufgrund von Ausfallzeiten durch automatisierte Hochverfügbarkeitsfunktionen und Service Level Agreements (SLAs) reduziert oder ganz vermieden werden. Weiterhin muss kaum noch eine Kapazitätsvorhersage vorgenommen werden, da Cloud-Datenbanken fast just in time erweitert werden können.
Niedrige Kosten	Durch das zeitweise Aktivieren und Deaktivieren von verschiedenen Services können die Kosten für den Datenbankbetrieb reduziert werden. Pay- Per-Use-Abonnementmodelle und dynamische Skalierung ermöglichen es, z.B. in Spitzenlastzeiten die Datenbankkapazitäten hoch zu skalieren und danach wieder auf ein normales Maß herunter zu skalieren. Dieses ist weitaus kostengünstiger, als müssten die Unternehmen physische Server erwerben, um die Lastspitzen zu bewältigen, obwohl dies möglicherweise nur einige Tage pro Quartal nötig ist.

Wenn sich Unternehmen wegen der Vorteile für eine Cloud-Datenbank entschieden haben, dann muss als Nächstes geklärt werden, wie sie diese verwalten möchten. Es gibt mehrere Varianten des Datenbankmanagements in der Cloud, welche allgemein in die folgenden vier Kategorien unterteilt werden können.



Cloud-Datenbankmanagement

Selbstverwaltete Cloud-Datenbank

In diesem Fall verlegt das Unternehmen seine Datenbank in die Cloud, managt diese jedoch selbst. Der Cloud-Anbieter stellt in diesem Fall nur die entsprechende Hardware zur Verfügung. Diese Form des Cloud-Datenbankmanagements ermöglicht eine enorme Anpassungsfähigkeit, Flexibilität und Funktionalität. Allerdings muss hierbei die Datenbank von einem Datenbankadministrator oder einem DevOps-Team manuell verwaltet werden.

Automatisierte Cloud-Datenbank

In dieser Variante verwenden Unternehmen bereitgestellte APIs für gewisse Datenbank-Cloud-Services. Sie haben jedoch den vollen Zugriff auf die Datenbankserver und kontrollieren die Datenbankkonfiguration und die Betriebssysteme. Automatisierte Datenbankservices weisen nur eingeschränkte SLA (Service Level Agreement) auf und schließen i.d.R. Aktivitäten wie Patching und Wartung aus.

Verwaltete Cloud-Datenbank

Diese Variante ist der Variante der automatisierten Cloud-Datenbanken sehr ähnlich. Allerdings hat der Kunde keinen Zugriff mehr auf die Server, auf denen die Datenbank gehostet wird. Der Kunde darf keine eigene Software installieren und eine Konfiguration der Datenbank ist nur im Rahmen der vom Cloud-Anbieter angebotenen Einstellungen möglich.

Autonome Cloud-Datenbank

Bei dieser Variante wird die menschliche Arbeit bezüglich des Datenbankmanagements und der Performanceoptimierung durch Automatisierung und Machine Learning auf ein Minimum reduziert bzw. vollständig abgeschafft. Zu den Services gehören u.a. SLA für geschäftskritische Anwendungen, z. B. die Ausfallsicherheit bei geplanten oder ungeplanten Datenbankaktivitäten.

Der neueste und innovativste Cloud-Datenbanktyp ist die autonome Cloud-Datenbank. Für die Verwaltung von Datenbanken in der Cloud ist normalerweise immer noch ein erfahrender Datenbankadministrator erforderlich. Dagegen erfordert die Verwaltung einer autonomen Cloud-Datenbank nicht mehr dieses umfassende Datenbank-Know-how eines Datenbankadministrators. Hier werden Cloud-Technologien und Machine Learning genutzt, um Verwaltungsaufgaben wie Datenbankoptimierung, Sicherheit, Backups, Aktualisierungen und andere Routineverwaltungsaufgaben zu automatisieren, welche bisher von Datenbankadministratoren ausgeführt werden. Autonome Datenbanken sind so konzipiert, dass sie Hardwarefehler automatisch erkennen und entsprechende Maßnahmen zum Ausgleich ergreifen. Außerdem skalieren sie je nach Bedarf die Performance und Kapazität der Cloud-Datenbank und schützen Daten vor externen und internen Angriffen.

Um den richtigen Cloud-Datenbankanbieter auszuwählen, sollten die Angebote verschiedener Anbieter miteinander verglichen werden. Zwei wichtige Vergleichskriterien sind dabei die Breite des Produktangebotes und die bereitgestellten Migrationsmechanismen. Durch eine breite Palette an DBaaS-Produkten ist es den Datenbankadministratoren möglich, eine optimale Datenbanklösung für ihr Unternehmen zusammenzustellen. Da in den meisten Unternehmen bereits große Datenbestände vorhanden sind, welche in eine Cloud-Datenbank migriert werden sollen, und dieser Prozess alles andere als einfach ist, ist es sehr wichtig, die Mechanismen, welche dafür von den einzelnen Anbietern zur Verfügung gestellt werden, miteinander zu vergleichen. Nachfolgend werden einige der größten Anbieter in diesem Bereich näher betrachtet.

Cloud-Datenbankanbieter

Amazon

Amazon besitzt eine breite Palette an DBaaS-Angeboten. Diese reichen von relationalen Datenbanken, z. B. SQL Server, Oracle, MySQL, PostgreSQL und Aurora bis hin zu NoSQL-Datenbanken wie DocumentDB, Neptune als Graphdatenbank und DynamoDB für Key-Value-Anwendungen. Redshift ist die Data-Warehouse-Plattform von Amazon, bei der u.a. Machine Learning eingesetzt wird, um Daten auszuwerten. Auch im Bereich der Migrations- und Konvertierungsmöglichkeiten ist Amazon sehr gut aufgestellt. Mit dem AWS Database Migration Service (DMS) besitzt Amazon ein leistungsstarkes Replikationsprogramm. Außerdem wird das Schema Conversion Tool angeboten. Damit lässt sich das Schema der Quelldatenbank und deren Views und Stored Procedures in die Amazon-Datenbank konvertieren. Das Produkt ermöglicht es zudem, Codes für Embedded-SQL-Anweisungen zu scannen und diese für den Zugriff auf das neue Datenbankprodukt verfügbar zu machen.

Microsoft

Auch Microsoft besitzt ein breites DBaaS-Produktportfolio. Zu den relationalen Produkten von Microsoft gehören sein Flaggschiff, die Azure SQL Database, aber auch MySQL, PostgreSQL, MariaDB sowie das SQL Data Warehouse für Big-Data-Anwendungen. Im Bereich der NoSQL-Datenbanken werden z. B. Cosmos DB, Azure Table Storage und Azure Cache for Redis angeboten. Für die Migration von Daten bietet Microsoft den Azure Database Migration Service an.

Google

Auch Google bietet einige DBaaS-Produkte an, wenn auch nicht so umfangreich wie Amazon oder Microsoft. Das Angebot von Google umfasst die relationale Datenbank Cloud Spanner und Cloud SQL, welche mit MySQL, PostgreSQL und SQL Server kompatibel ist. Zu den NoSQL-Produkten von Google gehören u.a. Google Cloud Bigtable, Cloud Firestore und Firebase Realtime Database. Auch bei den Migrationstools steht Google im Gegensatz zu Microsoft und Amazon erst am Anfang. Bei MySQL und PostgreSQL beispielsweise verlässt sich der Anbieter auf die integrierten Replikationsfunktionen der entsprechenden Datenbank. Außerdem empfiehlt Google bei größeren Problemen noch vielfach, auf andere Anbieter mit Erfahrung in den Bereichen Cloud-Migration und Datenbank-Konvertierung zurückzugreifen.

Oracle

Oracle legt seinen Schwerpunkt auf das Angebot von autonomen Datenbanken (S. 290), wie Autonomous Database und Autonomous Data Warehouse. Ergänzend werden aber auch mit MySQL relationale Datenbanken und mit Oracle NoSQL Database, Big Data Cloud Service und Oracle Exadata Cloud Service NoSQL-Datenbanken als DBaaS-Produkte angeboten. Werkzeuge für die heterogene Datenmigration werden von Oracle im Augenblick nicht zur Verfügung gestellt. Die einzige Unterstützung besteht in der Beschreibung von Methoden, wie lokale Oracle-Datenbanken in die Oracle Cloud zu konvertieren sind.

TAG 29 - Software testen und dokumentieren

Grundlage für dieses Kapitel bildet das Wissen über das Testen und die Dokumentation von Software. Die Inhalte umfassen den Testprozess, Testverfahren, Testarten, Teststufen und die verschiedenen Dokumentationsunterlagen. Aufbauend

auf diesem Wissen wird in diesem Abschnitt besonders auf Blackboxtest und den Einsatz von Testdatengeneratoren zur Erzeugung von Testdaten näher eingegangen. Auch das Wissen über die Dokumentation von Software wird vertieft. In diesem Zusammenhang wird speziell der Fokus auf das Abnahmeprotokoll gelegt.



LE 1 - Blackbox-Tests planen

Dynamische Testverfahren. werden eingesetzt, um die Funktionalität von Software zu testen. Diese lassen sich in Black-Box- und White-Box-Tests einteilen. Bei White-Box-Tests liegt der Quelltext vor und es wird versucht, eine möglichst hohe Testabdeckung zu erreichen. Deswegen ist ein White-Box-Test auch ein strukturorientiertes Testverfahren. Der Black-Box-Test hingegen ist ein spezifikationsorientiertes Verfahren, weil hier nicht der Quellcode, sondern nur die Schnittstelle und deren Anforderungsspezifikationen getestet werden.



White-Box-Tests haben den Nachteil, dass sie sehr stark an den Quellcode gekoppelt sind. Das heißt, bei Optimierung, Erweiterung oder einfach nur bei einigen Änderungen am Quellcode verlieren sie ihre Gültigkeit, da man sich nicht sicher sein kann, ob die komplette Anweisungs- oder Entscheidungsüberdeckung noch gegeben ist. Die Tests müssen also jedes Mal überprüft und gegebenenfalls angepasst werden. Black-Box-Tests hingegen weisen diesen Nachteil nicht auf. Sie können, solange keine Änderungen in der Anforderungsspezifikation vorliegen, immer wieder zum Testen des Programms verwendet werden. Hier stellt sich allerdings die Frage, welche Testdaten verwendet werden sollen. Zur Ermittlung von geeigneten Testdaten gibt es z. B. die Verfahren Äquivalenzklassenbildung und Grenzwertanalyse.

Verfahren	Beschreibung											
Äquivalenzklassenbildung	Mit diesem Verfahren wird versucht, eine möglichst hohe Testabdeckung mit sehr wenigen Tests zu erreichen. Dabei werden die Testfälle in Äquivalenzklassen eingeteilt. Es werden gültige und ungültige Äquivalenzklassen unterschieden. Bei gültigen Äquivalenzklassen werden gültige Testdaten verwendet und bei ungültigen Äquivalenzklassen ungültige Testdaten. Die einzelnen Testfälle zu den Äquivalenzklassen werden wie folgt gebildet: - Analyse und Spezifikation der Eingabe- und der Ausgabedaten entsprechend den Anforderungsspezifikationen, - Bildung der Äquivalenzklassen durch Klassifizierung der Wertebereiche für die Ein- und Ausgabedaten, - Bestimmung der Testfälle durch Werteauswahl für jede Äquivalenzklasse.											
	Beispiel Eine Methode „bool IsTemperatureOk (double temp)“ soll <i>True</i> zurückgegeben, wenn der übergebene Temperaturwert kleiner gleich 30 °C ist. Über 30 °C soll <i>False</i> zurückgegeben werden. Im Beispiel gibt es zwei Äquivalenzklassen: 1. Temperatur <= 30 und 2. Temperatur > 30. Für beide Äquivalenzklassen gilt, dass sie gültige Äquivalenzklassen darstellen und sehr viele Testwerte enthalten. Ungültige Äquivalenzklassen, z. B. Temperatureingaben unter dem absoluten Nullpunkt, sollen in dieser Funktion nicht berücksichtigt werden. Beim Testen würde z. B. ein Testwert für die 1. Äquivalenzklasse von -1 °C den gleichen Effekt haben wie ein Testwert von 28 °C. Das gleiche gilt für die zweite Äquivalenzklasse. Das Testen aller Werte einer Äquivalenzklasse kann sehr aufwendig oder - wie in diesem Fall - sogar unmöglich sein. Deswegen werden aus jeder Äquivalenzklasse entsprechende Werte ausgewählt. <table><tr><td>Äquivalenzklasse</td><td>1</td><td>2</td></tr><tr><td>Wertebereich</td><td>Temperatur <= 30</td><td>Temperatur > 30</td></tr><tr><td>Mögliche Testwerte</td><td>-1, 3.89, -12.99, 28</td><td>32, 101.33, 65.4, 40</td></tr><tr><td>Erwartetes Ergebnis</td><td>True</td><td>False</td></tr></table>	Äquivalenzklasse	1	2	Wertebereich	Temperatur <= 30	Temperatur > 30	Mögliche Testwerte	-1, 3.89, -12.99, 28	32, 101.33, 65.4, 40	Erwartetes Ergebnis	True
Äquivalenzklasse	1	2										
Wertebereich	Temperatur <= 30	Temperatur > 30										
Mögliche Testwerte	-1, 3.89, -12.99, 28	32, 101.33, 65.4, 40										
Erwartetes Ergebnis	True	False										
Grenzwertanalyse	Die Grenzwertanalyse stellt eine Erweiterung der Äquivalenzklassenbildung dar. In vielen Fällen treten die Fehler in den Programmen hauptsächlich an Grenzen der entsprechenden Äquivalenzklasse auf, z. B. durch das Setzen falscher Vergleichsoperatoren (< anstatt <=). Deswegen werden bei der Grenzwertanalyse Testdaten ausgewählt, welche die Grenzen einer Äquivalenzklasse und damit einen Übergang zu einer anderen Äquivalenzklasse darstellen.											
	Beispiel Eine Methode „bool IsTemperatureOk (double temp)“ soll <i>True</i> zurückgegeben, wenn der übergebene Temperaturwert kleiner gleich 30 °C ist. Über 30 °C soll sie <i>False</i> zurückgeben. Wie schon bei der Äquivalenzklassenbildung können hier zwei Äquivalenzklassen gebildet werden. Auch hier spielt zum Testen der Funktion der absolute Nullpunkt keine Rolle. Es sollen der Funktion nur Werte mit zwei Nachkommastellen übergeben werden. Aus diesen Angaben ergeben sich die folgenden Testfälle: <table><tr><td>Äquivalenzklasse</td><td>1</td><td>2</td></tr><tr><td>Wertebereich</td><td>Temperatur <= 30</td><td>Temperatur > 30</td></tr><tr><td>Testwerte</td><td>30.00</td><td>30.01</td></tr><tr><td>Erwartetes Ergebnis</td><td>True</td><td>False</td></tr></table>	Äquivalenzklasse	1	2	Wertebereich	Temperatur <= 30	Temperatur > 30	Testwerte	30.00	30.01	Erwartetes Ergebnis	True
Äquivalenzklasse	1	2										
Wertebereich	Temperatur <= 30	Temperatur > 30										
Testwerte	30.00	30.01										
Erwartetes Ergebnis	True	False										

Eine weitere Möglichkeit, um Testdaten zu gewinnen, besteht in der Nutzung von Entscheidungstabellen. Hierbei werden die Anforderungsspezifikationen mithilfe einer Entscheidungstabelle abgebildet. Jede Spalte einer Entscheidungstabelle entspricht dabei einem Testfall. Die Erstellung von Entscheidungstabellen ist leicht und kann durch einfache Regeln durchgeführt werden. Dieses gilt vor allem für das Testen von einfacher Logik. Für komplexe Logik können Entscheidungstabellen allerdings schnell unübersichtlich werden.

Entscheidungstabelle	Regeln			
Bedingungen	R1	R2	R3	R4
Sensor 1 sendet Signal	J	J	N	N
Sensor 2 sendet Signal	J	N	J	N
Aktionen				
Alarm auslösen	x	-	-	-
Warnmeldung anzeigen	-	x	x	-

LE 2 - Testdatengeneratoren verwenden

Mithilfe von Testdatengeneratoren werden Daten erzeugt, die u.a. zur Ausführung von Testfällen verwendet werden können. Diese Tools kommen meistens dann zum Einsatz, wenn entsprechende Testdaten nicht oder nicht in ausreichender Menge verfügbar sind. Die besten Testdatengeneratoren helfen bei der Durchführung von Last-, Leistungs-, Stresstests. Solche Tests wären ohne Testdatengeneratoren teilweise gar nicht möglich. Mit diesen Tools generierte Daten können auch in Datenbanken zum Testen verwendet werden. Das geht sogar so weit, dass sie vertrauliche Daten durch Dummy-Daten ersetzen und gleichzeitig die vertraulichen Daten aufbewahren.



Testdatengenerator

Ein Testdatengenerator ist ein Unterstützungswerkzeug beim Testen von Anwendungen, mit dem die für einen Test benötigten Daten generiert, bereitgestellt, verändert oder aus einer Wissensdatenbank selektiert werden können.

Dadurch, dass mithilfe von Testdatengeneratoren ein großes Datenvolumen sehr schnell zu erzeugen ist, sind diese auch sehr wichtig zum Testen von Big-Data-Anwendungen. Testdatengeneratoren lassen sich in verschiedene Kategorien einteilen, wobei manche auch mehrere Kategorien bedienen.

Kategorie	Beschreibung
Datenbankbasierte Testdatengeneratoren	Sie verarbeiten Datenbankschemata und sind in der Lage, daraus Testdatenbestände zu generieren.
Codebasierte Testdatengeneratoren	Sie generieren die Testdaten, indem sie den Code des Testobjekts analysieren (White-Box-Test).
Schnittstellenbasierte Testdatengeneratoren	Sie analysieren die Testobjektschnittstelle, erkennen die Definitionsbereiche der Schnittstellenparameter und leiten, z. B. mittels Äquivalenzklassen- und Grenzwertanalyse, daraus Testdaten ab (Black-Box-Test).
Spezifikationsbasierte Testdatengeneratoren	Sie leiten Testdaten und zugehörige Sollwerte aus einer Spezifikation ab.

Auf dem Markt sind sehr viele Tools zur Generierung von Testdaten verfügbar. Die Bandbreite reicht von einfachen, frei verfügbaren Online-Tools für die Generierung von Daten im CSV-Format bis hin zu hoch spezialisierten Generatoren für den Big-Data-Bereich. Bei der Auswahl eines geeigneten Tools müssen jedoch verschiedene Faktoren berücksichtigt werden, z. B.

die unterstützten Datenbanken und Datenformate, die Datengenerierungsmethoden, die Betriebssystemunterstützung, die Kosten und die Verfügbarkeit. Im Folgenden sind einige viel genutzte Anwendungen zur Testdatengenerierung aufgeführt.

Anwendung	Eigenschaften	Datenbank- und Formatunterstützung	Kosten
DATPROF	intuitiv und einfach zu bedienen www.datprof.com	u.a. Oracle, MS SQLServer, PostgreSQL, MySQL, MariaDB	für privaten Gebrauch kostenlos nutzbar
GenerateDataxom	einfach zu verwenden, anpassbare Datentypen, Länder-Plug-In www.generate-data.com	MYSQL	kostenlos nutzbar
IRI RowGen	NID- und E-Mail-Generatoren, Datenklassen- und Regelbibliotheken, integrierte Datentransformation und Berichtsformatierung von Testdaten www.iri.com/products/rowgen	jede relationale Datenbank mit JDBC connection; viele NoSQL-Datenbanken	teilweise kostenlos nutzbar
Mockaroo	erzeugt realistische Daten, unterstützt mehr als 100 Datentypen www.mockaroo.com	SQL-, CSV-, JSON-, Excelformat	kostenlos nutzbar
Datanamic Data Generator MultiDB	Erweiterte Einstellungen für die Datengenerierung und -Validierung www.datanamic.com/datagenerator	Oracle, MS SQL-Server, MySQL, PostgreSQL, MS Access, SQLite	kostenpflichtig
GS Data Generator	unterstützt die Systemintegration www.testtoolreview.de/de/testing-tools/tool-liste/tool-detail/406-gs-data-generator	MS SQL-Server, Oracle, DB2, MS Access, Fox Pro Excelformat, CSV-Format	kostenlos nutzbar

Aus dieser Liste wird der Testdatengenerator Mockaroo beispielhaft etwas genauer betrachtet: Der Testdatengenerator unterstützt mehr als einhundert Datentypen, ist leicht zu bedienen und bietet Unterstützung beim Generieren von Daten in den Formaten CSV, JSON, SQL und Excel. Es werden sehr realistische Daten erstellt und für geografische Felder wie Land, Stadt usw. wird auf reale, zusammenhängende Daten zurückgegriffen.

Dieser Fall wird hier einmal exemplarisch dargestellt. Es sollen zur Anschauung zehn Testdatensätze für Länder erzeugt werden, welche folgende Daten pro Datensatz enthalten: ID, Name des Landes, Ländercode, Währung, Währungscode, Zeitzone. Danach werden Testdaten sowohl im CSV-, JSON- und XML-Format erzeugt. Diese kann man sich als Vorschau anzeigen lassen und danach in dem entsprechenden Format speichern. Die Einstellungen und erzeugten Daten in verschiedenen Formaten können wie folgt aussehen:

Field Name	Type	Options
id	Row Number	blank: 0% Σ X
country	Country	restrict countries...
country code	Country Code	blank: 0% Σ X
currency	Currency	blank: 0% Σ X
currency code	Currency Code	blank: 0% Σ X
time zone	Time Zone	blank: 0% Σ X

ADD ANOTHER FIELD

Rows: 10 Format: CSV Line Ending: Unix (LF) Include: ☒ header ☐ BOM

Einstellungen in der Oberfläche

Preview

```
[
  {
    "id": 1,
    "country": "United States",
    "country code": "US",
    "currency": "Dollar",
    "currency code": "USD",
    "time zone": "America/New_York"
  },
  {
    "id": 2,
    "country": "Indonesia",
    "country code": "ID",
    "currency": "Rupiah",
    "currency code": "IDR",
    "time zone": "Asia/Makassar"
  },
  {
    "id": 3,
    "country": "Iran",
    "country code": "IR",
    "currency": "Rial",
    "currency code": "IRR",
    "time zone": "Asia/Tehran"
  },
  {
    "id": 4,
    "country": "Canada",
    "country code": "CA",
    "currency": "Dollar",
    "currency code": "CAD",
    "time zone": "America/Vancouver"
  },
  {
    "id": 5,
    "country": "Indonesia",
    "country code": "ID",
    "currency": "Rupiah",
    "currency code": "IDR",
    "time zone": "Asia/Jakarta"
  }
]
```

Testdaten im JSON-Format (siehe S. 173)

Preview

```
<?xml version="1.0" encoding="UTF-8"?>
<dataset>
  <record>
    <id>1</id>
    <country>South Korea</country>
    <country code>KR</country code>
    <currency>Won</currency>
    <currency code>KRW</currency code>
    <time zone>Asia/Seoul</time zone>
  </record>
  <record>
    <id>2</id>
    <country>Ireland</country>
    <country code>IE</country code>
    <currency>Euro</currency>
    <currency code>EUR</currency code>
    <time zone>Europe/Dublin</time zone>
  </record>
  <record>
    <id>3</id>
    <country>Russia</country>
    <country code>RU</country code>
    <currency>Ruble</currency>
    <currency code>RUB</currency code>
    <time zone>Europe/Moscow</time zone>
  </record>
  <record>
    <id>4</id>
    <country>Indonesia</country>
    <country code>ID</country code>
    <currency>Rupiah</currency>
    <currency code>IDR</currency code>
    <time zone>Asia/Makassar</time zone>
  </record>

```

Testdaten im XML-Format (siehe S. 172)

Preview

```
id,country,country code,currency,currency code,time zone
1,Macedonia,MK,Denar,MED,Europe/Skopje
2,Brazil,BR,Real,BRL,America/Sao_Paulo
3,France,FR,Euro,EUR,Europe/Paris
4,Iran,IR,Rial,IRR,Asia/Tehran
5,Poland,PL,Zloty,PLN,Europe/Warsaw
6,China,CN,Yuan_Renminbi,CNY,Asia/Shanghai
7,Panama,PA,Balboa,PAB,America/Panama
8,Mozambique,MZ,Meticul,MZN,Africa/Maputo
9,Russia,RU,Ruble,RUB,Europe/Moscow
10,Norway,NO,Krone,NOK,Europe/Oslo
```

Landesday - Editor

```
Datei Bearbeiten Format Ansicht Hilfe
id,country,country code,currency,currency code,time zone
1,Sweden,SE,Krona,SEK,Europe/Stockholm
2,Greece,GR,Euro,EUR,Europe/Athens
3,China,CN,Yuan_Renminbi,CNY,Asia/Shanghai
4,Nigeria,NG,Naira,NGN,Africa/Lagos
5,Serbia,RS,Dinar,RSD,Europe/Belgrade
6,Czech_Republic,CZ,Koruna,CZK,Europe/Prague
7,EUR,Europe/Paris
8,EUR,Europe/Dublin
9,Rupiah,IDR,Asia/Jakarta
10,Peso,PHP,Asia/Manila
```

Ze 1, Sp 1 100% Unix (LF) UTF-8

Testdaten im CSV-Format; als Vorschau und als CSV-Datei gespeichert

Die abgespeicherten Daten können nun zur Testzwecken weiter genutzt werden. In der Regel werden dafür mehr als zehn Datensätze erzeugt.

LE 3 - Projektabnahmen durchführen und Abnahmeprotokolle erstellen

Wenn ein Softwareprodukt fertiggestellt ist, dann erfolgt i.d.R. die Abnahme. Besteht ein Werkvertrag zwischen Auftraggeber und Auftragnehmer, dann ist eine solche Abnahme sogar gesetzlich vorgeschrieben. Bei der Abnahme beurteilt der Auftraggeber die Ergebnisse der Entwicklung und nimmt das Softwareprodukt entgegen. Außerdem muss der Auftraggeber erklären, dass das Softwareprodukt als vertragsgemäße Leistung im Wesentlichen anerkannt wird.

Abnahme nach DIN 69901-5:2009-1

Eine Abnahme ist die Bestätigung des Auftraggebers, dass die Anforderungen durch den Auftragnehmer erfüllt wurden.

Die Abnahme ist deshalb von entscheidender Bedeutung, weil damit bestimmte Rechtsfolgen für das IT- Projekt verbunden sind. Einige davon wären:

- Fälligkeit des Zahlungsanspruches des Auftragnehmers,
- Entstehung möglicher Mängelgewährleistungspflichten,
- Verjährungsbeginn für die Mängelrechte,
- Beweislast für Mängel trifft ab diesem Zeitpunkt den Kunden usw.

Die Abnahme als solche basiert auf den im Vertrag oder im Pflichtenheft getroffenen Vereinbarungen und endet mit der Akzeptanz verschiedener Aspekte.

Aspekte der Abnahme	Beispielhafte Inhalte
Vertragliche Akzeptanz	• Vollständigkeit der vereinbarten Liefergegenstände und Leistungen • Qualität der Liefergegenstände und Leistungen • Erfüllung der gesetzlichen Anforderungen
Benutzerakzeptanz	• Erfüllung der funktionalen Anforderungen (z. B. Ergebnisse eines Testbetriebs) • Erfüllung nichtfunktionaler Anforderungen (z. B. Korrektheit und Verständlichkeit der Dokumentation und Handbücher oder Ergebnisse aus Performance-Tests)
Akzeptanz durch den zukünftigen Betreiber des IT-Systems	• Erfüllung der technischen Anforderungen • Erfüllung der Anforderungen an Daten- und Revisionssicherheit • Erfüllung der IT-Sicherheitsanforderungen • Vollständigkeit und Qualität der Admin-Handbücher

Die Abnahme als solche wird in einem Abnahmeprotokoll festgehalten und von den Personen unterschrieben, welche die Abnahme durchführen. Das sind i.d.R. der Auftraggeber und der Auftragnehmer oder von ihnen autorisierte Personen. Das Abnahmeprotokoll dokumentiert den Ablauf und die Ergebnisse der Abnahme und ist die Grundlage dafür, um das Softwareprodukt vollständig anzunehmen oder Nachbesserungsmaßnahmen einzufordern. Die derzeit gültigen Projektmanagementstandards enthalten keine bestimmte Vorgabe für ein solches Abnahmeprotokoll. Der PMBOK-Standard benennt beispielsweise überhaupt kein Dokument, das die Abnahme protokolliert. Im PRINCEZ-Standard wird die Abnahmedokumentation (acceptance record) zwar erwähnt, aber nicht näher beschrieben. So ist es i.d.R. jedem Unternehmen selbst überlassen, wie das Abnahmeprotokoll aussehen soll. Als Minimum sollte zumindest eine Checkliste mit den festgelegten Abnahmekriterien im Abnahmeprotokoll enthalten sein. Empfehlenswert ist es aber, folgende Punkte in das Abnahmeprotokoll mit aufzunehmen:

Inhalte, welche im Abnahmeprotokoll enthalten sein sollten

- Ort, Datum, Uhrzeit der Abnahme
- Teilnehmer und ihre Funktion im Projekt und bei der Prüfung
- Genaue Bezeichnung der Software-Version und Details zur geprüften Systemumgebung
- Geprüfte Eigenschaften und deren Bezug zu den Abnahmekriterien
- Art der Prüfungen
- Ergebnisse der Prüfungen
- Nicht geprüfte Abnahmekriterien
- Beurteilung, ob der Vertrag erfüllt ist
- Offene Punkte, auf deren Erfüllung der Auftraggeber verzichtet
- Offene Punkte, welche noch zu erfüllen sind (mit Termin und Verantwortlichkeit)
- Weitere Vereinbarungen bzw. Prüfungen

Darüber hinaus sollte eine Aussage über die Vollständigkeit, Funktionseignung und Schadensfreiheit der geprüften Software getroffen werden.

Ein einfaches Abnahmeprotokoll könnte wie folgt gestaltet sein.

Abnahmeprotokoll		
Teilnehmer/-innen:		Datum: 21.05.2021
Name	Firma	Funktion
Schulz, Maria	JIKU IT-Solutions GmbH	Projektleiterin
Müller, Karla	ISOTO GmbH	Geschäftsführerin
1. Software: Statistika Version 1.0 2. Vertragsnummer: 71-U2020 3. Ergebnisse der Annahmeprüfung:		
Funktionale Anforderungen		
Abnahmekriterium	Ergebnis der Prüfung	
Vier grafische Darstellungsmöglichkeiten der statistischen Auswertung	Ist vollständig erfüllt	
...	...	
Nichtfunktionale Anforderungen		
Abnahmekriterium	Ergebnis der Prüfung	
Umfangreiche und leicht zu verstehende Bedienungsanleitung	Ist vollständig vorhanden	
...	...	
Technische Anforderungen		
Abnahmekriterium	Ergebnis der Prüfung	
Software soll unter Windows 10 und Linux ausführbar sein	Läuft sowohl unter Windows 10 als auch Linux	
...	...	
4. Offene Punkte, welche noch zu erfüllen sind: Der Auftragnehmer hat noch folgende Aufgaben im Rahmen des Vertrages zu erfüllen: <i>keine</i>		
Aufgabe	Termin	
...	...	

TAG 30 - Vertiefung des Wissens und Modulprüfung Teil 2

LE 1 - Einleitung

Der letzte Tag in diesem Modul bietet Ihnen die Möglichkeit der Wiederholung und Vertiefung Ihres Wissens. Sollten es noch offene, nicht vollständig bearbeitete Modul-Inhalte geben, nutzen Sie den heutigen Tag zur Komplettierung.

LE 2 - Modulprüfung

120 Minuten zu denen wir Ihnen recht viel Erfolg wünschen!