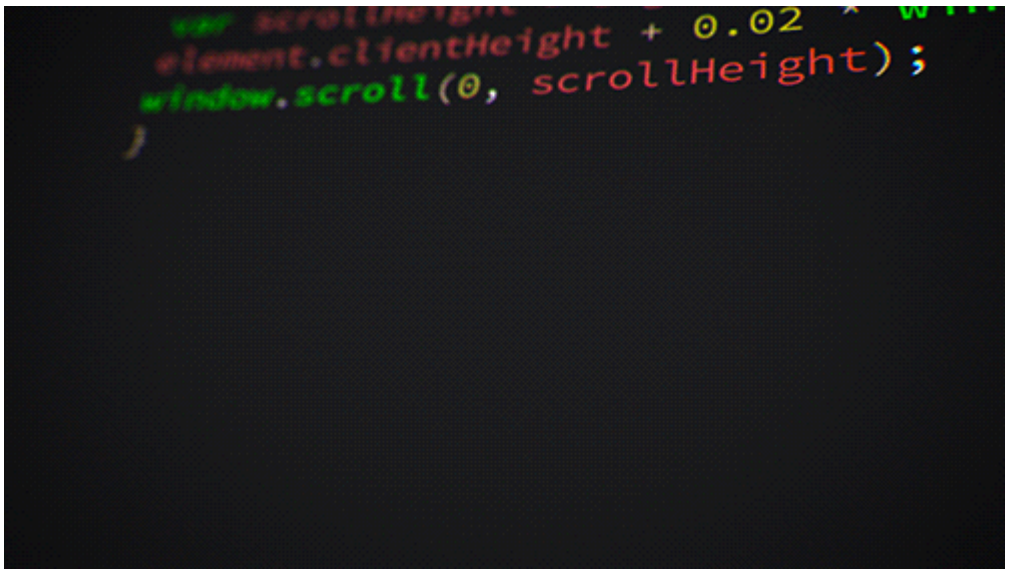
Modellierung von Software

Parameter	Kursinformationen
Veranstaltung:	`Vorlesung Softwareentwicklung`
Teil:	`14/27`
Semester	`Sommersemester 2025`
Hochschule:	`Technische Universität Freiberg`
Inhalte:	Klassifikation von UML Diagrammtypen, Anwendungsfall Diagramm, Aktivitätsdiagramm, Sequenzdiagramm, Klassendiagramm, Objektdiagramm, UML Tools
Link auf den GitHub:	https://github.com/TUBAF-IfI-LiaScript/VL_Softwareentwicklung/blob/master/16_UML_ModellierungII.md
Autoren	Sebastian Zug, Galina Rudolf & André Dietrich



Warum die Formalisierung?

Lastenheft und Pflichtenheft:

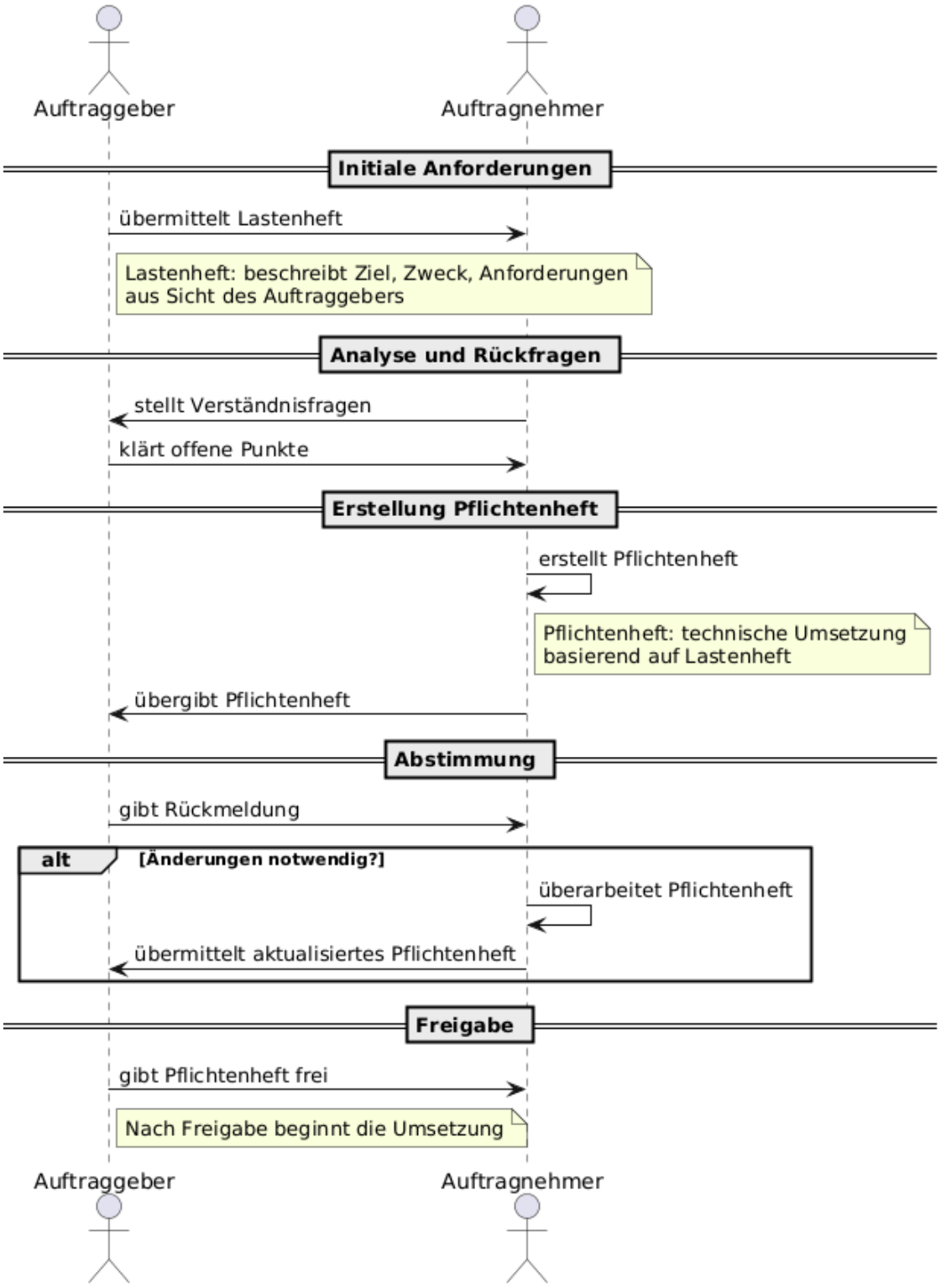
1. Das Lastenheft beschreibt alle Anforderungen und Wünsche des Auftraggebers an ein zukünftiges System, u.a. *funktionale Anforderungen*: was soll das System tun? (Features, Anwendungsfälle).

   Beispiel aus dem Lastenheft (was? — Kundensicht): *Der Webshop soll es Endkunden ermöglichen, Produkte aus dem Bereich Haushaltswaren online zu bestellen. Ziel ist es, einen nutzerfreundlichen, performanten und responsiven Online-Shop bereitzustellen, der eine einfache Produktsuche, einen sicheren Bestellprozess sowie gängige Zahlungsmethoden (z. B. PayPal, Kreditkarte) unterstützt.*

2. Das Pflichtenheft beschreibt, wie die Anforderungen des Lastenhefts umgesetzt werden sollen, d.h. es enthält detaillierte Spezifikationen und Entwürfe für die Realisierung des Systems und enthält ebenfalls einen Abschnitt zu funktionalen Anforderungen.

   Beispiel aus dem Pflichtenheft (wie? — Entwicklersicht): *Der Webshop wird mit dem Framework Django umgesetzt und verwendet eine PostgreSQL-Datenbank. Die Produktsuche wird über ein Volltext-Suchfeld mit Autovervollständigung realisiert. Der Bestellprozess besteht aus folgenden Schritten:*

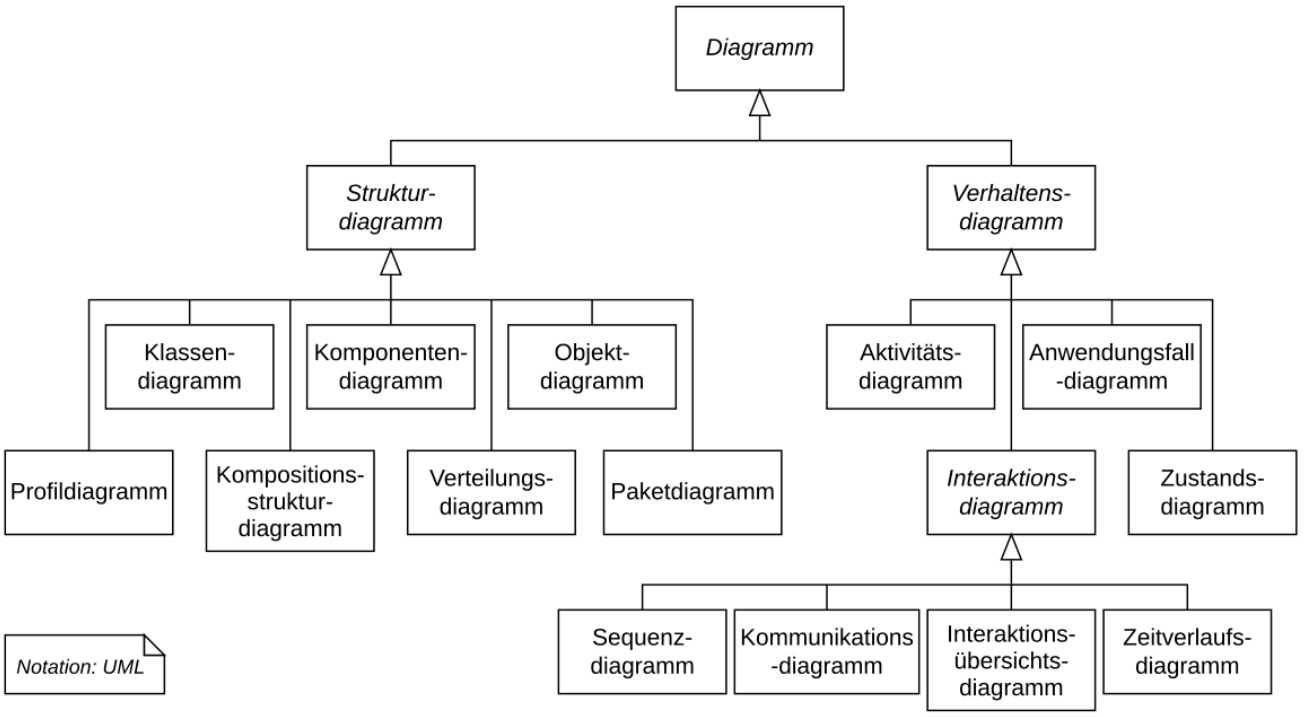
   - Warenkorb anzeigen und bearbeiten
   - Benutzer-Login oder Gastbestellung
   - Eingabe der Lieferadresse
   - Auswahl der Zahlungsart (Stripe-Integration für Kreditkarte und PayPal)
   - Bestellübersicht mit Bestätigungsfunktion
   - Automatischer E-Mail-Versand der Bestellbestätigung



Merke: Das Lastenheft beschreibt die Anforderungen aus Sicht des Auftraggebers, während das Pflichtenheft die technische Umsetzung dieser Anforderungen aus Sicht des Auftragnehmers beschreibt.

Merke: UML-Diagramme sind ein wichtiges Hilfsmittel, um die Anforderungen und das Design eines Systems zu visualisieren und zu kommunizieren. Sie helfen dabei, komplexe Zusammenhänge verständlich darzustellen und dienen als Grundlage für die Implementierung.

UML Diagrammtypen



Autor Stkl - derivative work: File: UML-Diagrammhierarchie.png: Sae1962, CC BY-SA 4.0, https://upload.wikimedia.org/wikipedia/commons/thumb/d/da/UML-Diagrammhierarchie.svg/1200px-UML-Diagrammhierarchie.svg.png

OMG-Spezifikation

Im folgenden werden wir uns aus den beiden Hauptkategorien jeweils folgende Diagrammtypen genauer anschauen:

- Verhaltensdiagramme
  - Anwendungsfall Diagramm
  - Aktivitätsdiagramm
  - Sequenzdiagramm
- Strukturdiagramme
  - Klassendiagramm
  - Objektdiagramm

Anwendungsfall Diagramm

> Das Anwendungsfalldiagramm (Use-Case Diagramm) abstrahiert das erwartete Verhalten eines Systems und wird dafür eingesetzt, die Anforderungen an ein System zu spezifizieren.

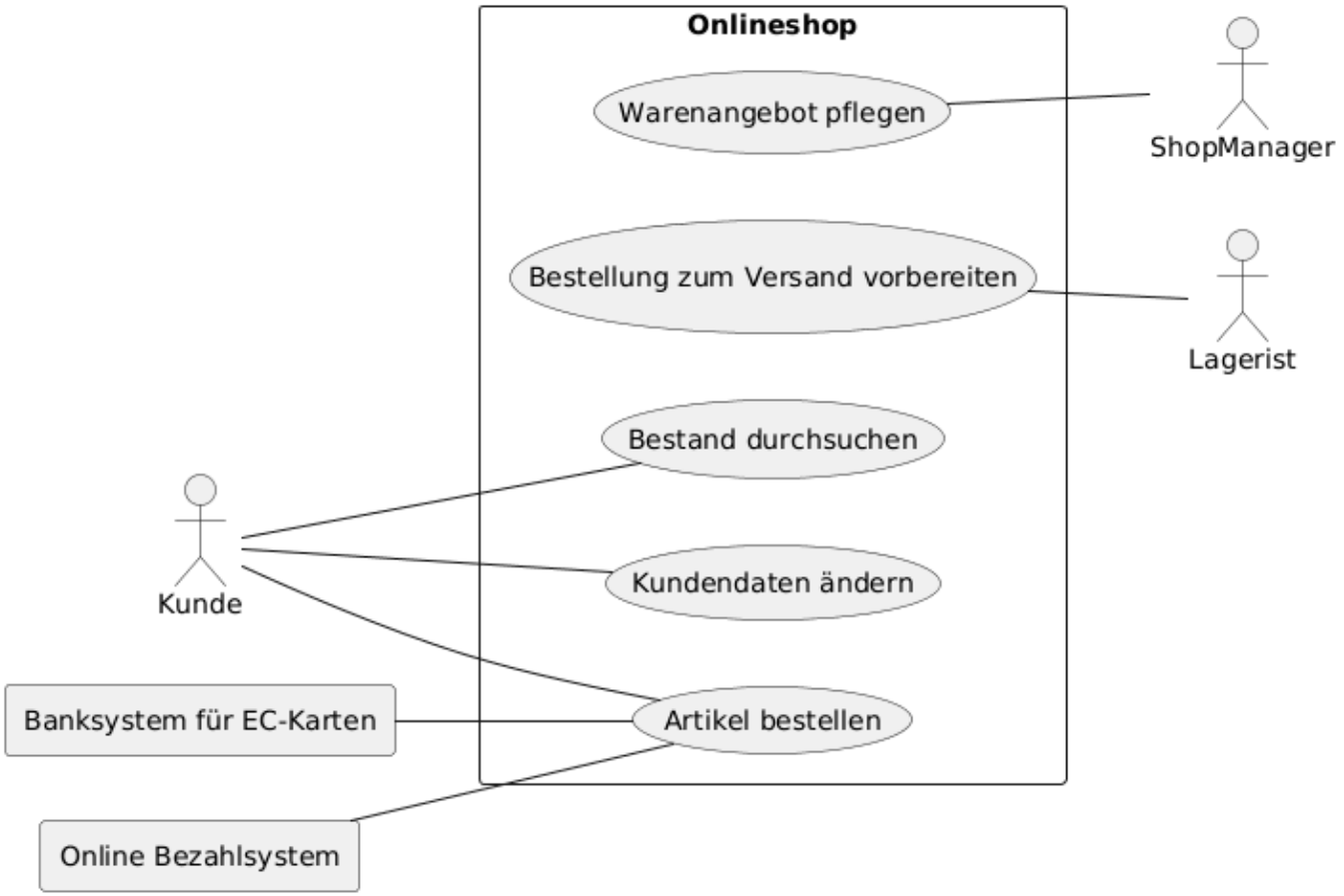
Ein Anwendungsfalldiagramm ist eine grafische Darstellung der *funktionalen Anforderungen* eines Systems. Es zeigt die verschiedenen Anwendungsfälle (Use Cases), Akteure und deren Interaktionen.

> Achtung: Ein Anwendungsfalldiagramm stellt keine Ablaufbeschreibung dar! Diese kann stattdessen mit einem Aktivitäts-, einem Sequenz- oder einem Kommunikationsdiagramm dargestellt werden.

Basiskonzepte

Elemente:

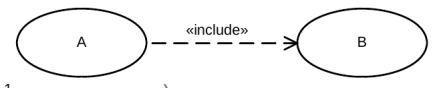
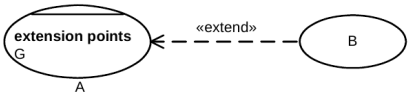
- Systemgrenzen werden durch Rechtecke gekennzeichnet.
- Akteure werden als „Strichmännchen“ dargestellt, dies können sowohl Personen (Kunden, Administratoren) als auch technische Systeme sein (manchmal auch ein Bandsymbol verwendet). Sie ordnen den Symbolen Rollen zu.
- Anwendungsfälle werden in Ellipsen dargestellt. Üblich ist die Kombination aus Verb und ein Substantiv `Kundendaten Ändern`.
- Beziehungen zwischen Akteuren und Anwendungsfällen müssen durch Linien gekennzeichnet werden. Man unterscheidet "Association", "Include", "Extend" und "Generalization".

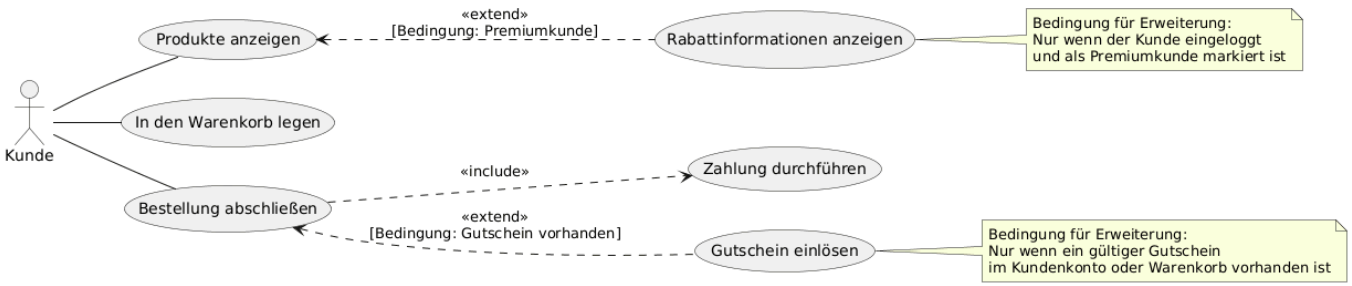


Einfaches Anwendungsfalldiagramm für einen Online-Versand

Verfeinerung

Use-Case Diagramme erlauben die Abstraktion von Elementen auf der Basis von Generalisierungen. So können Akteure von einander erben und redundante Beschreibungen von Verhalten über `<<extend>>` oder `<<include>>` (unter bestimmten Bedingungen) erweitert werden.

		
	`<<include>>` Beziehung	`<<extend>>` Beziehung
Bedeutung	Ablauf von A schließt den Ablauf von B immer ein	Ablauf von A kann optional um B erweitert werden
Anwendung	Hierachische Zerlegung	Abbildung von Sonderfällen
Abhängigkeiten	A muss B bei der Modellierung berücksichtigen	Unabhängige Modellierung möglich



Anwendungsfälle

- Darstellung der wichtigsten Systemfunktionen
- Austausch mit dem Anwender und dem Management auf der Basis logischer, handhabbarer Teile
- Dokumentation des Systemüberblicks und der Außenschnittstellen
- Indentifikation von Anwendungsfällen

Vermeiden Sie ...

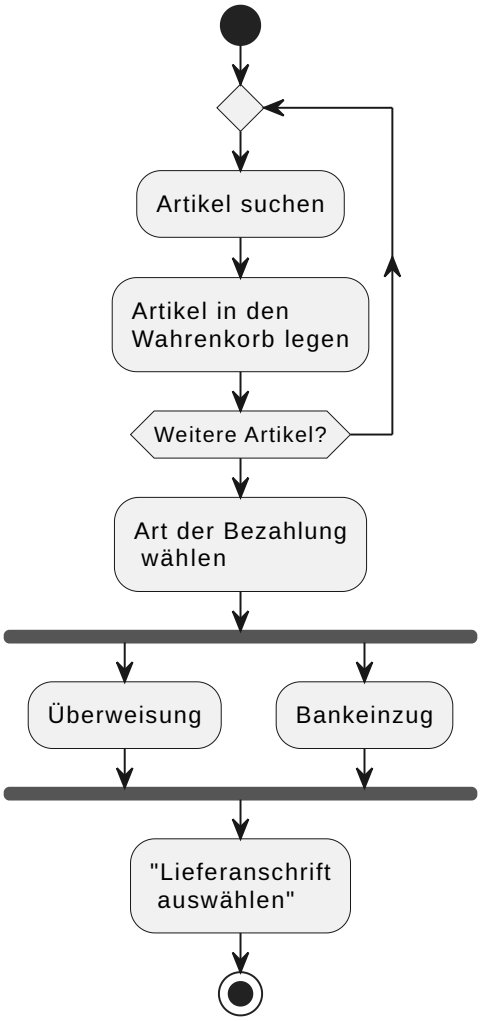
- ... eine zu detaillierte Beschreibung von Operationen und Funktionen
- ... nicht funktionale Anforderungen mit einem Use-Case abbilden zu wollen
- ... Use-Case Analysen aus Entwicklersicht durchzuführen
- ... zu viele Use-Cases in einem Diagramm abzubilden (max. 10)

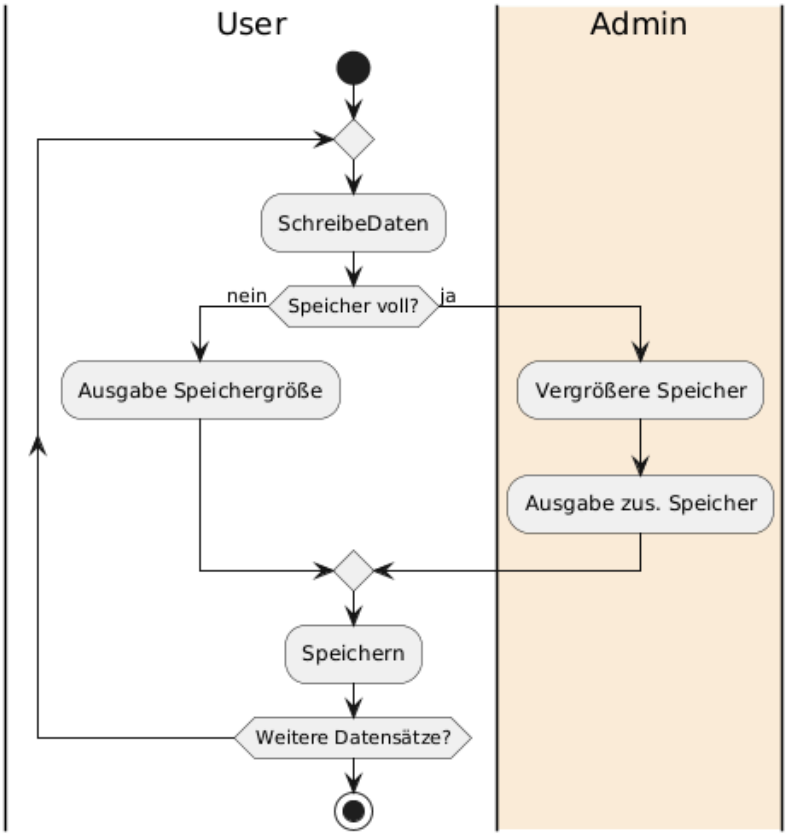
> Halten Sie komplexe `<extend>` Bedingungen in separaten Dokumenten fest.

Aktivitätsdiagramm

> Aktivitätsdiagramme stellen die Vernetzung von elementaren Aktionen und deren Verbindungen mit Kontroll- und Datenflüssen grafisch dar.

Aktivitätsmodellierung in UML1





Aktivitätsdiagramme.plantUML	ActivityUser.plantUML

Elemente UML 1.x:

- Aktivitätszustände (Activity States)
- Übergänge (Transitions)
- Startzustand (Initial State)
- Endzustand (Final State)
- Entscheidungsknoten (Decision Nodes)
- Synchronisationsbalken
- Aktivitätspartitionen.

Bis UML 1.x waren Aktivitätsdiagramme eine Mischung aus Zustandsdiagramm, Petrinetz und Ereignisdiagramm, was zu theoretischen und praktischen Problemen führte.
Erweiterung des Konzeptes in UML2

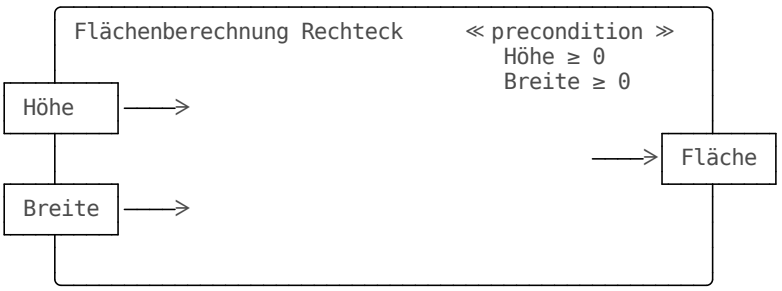
Bereich	UML 1.x	UML 2.x
Strukturmodell	einfaches Kontrollflussmodell (ähnlich Flussdiagramm)	basiert auf **aktivitätsbasiertem Zustandsautomaten** mit Tokens (ähnlich **Petri-Netzen**)
Objektfluss	rudimentär	**explizite Darstellung von Datenflüssen** zwischen Aktionen möglich
Pins	nicht vorhanden	**Input- und Output-Pins** ermöglichen feingranulare Kontrolle über Daten
Partitions (Swimlanes)	einfach	weiterhin vorhanden, aber besser formalisiert
Unteraktivitäten	eingeschränkt	**Call Behavior Action** für Wiederverwendung anderer Aktivitäten
Nebenläufigkeit (Concurrency)	schwer modellierbar	durch **Fork/Join-Nodes** sauber formalisiert
Signale und Ereignisse	kaum genutzt	erweiterte Unterstützung für **Signal Send/Receive Actions**
Exception Handling	nicht vorhanden	neu eingeführt mit **Interruptible Regions** und **Exception Handlers**
Zustandsübergänge	implizit	explizite Modellierung möglich

> "Was früher Aktivitäten waren sind heute Aktionen."

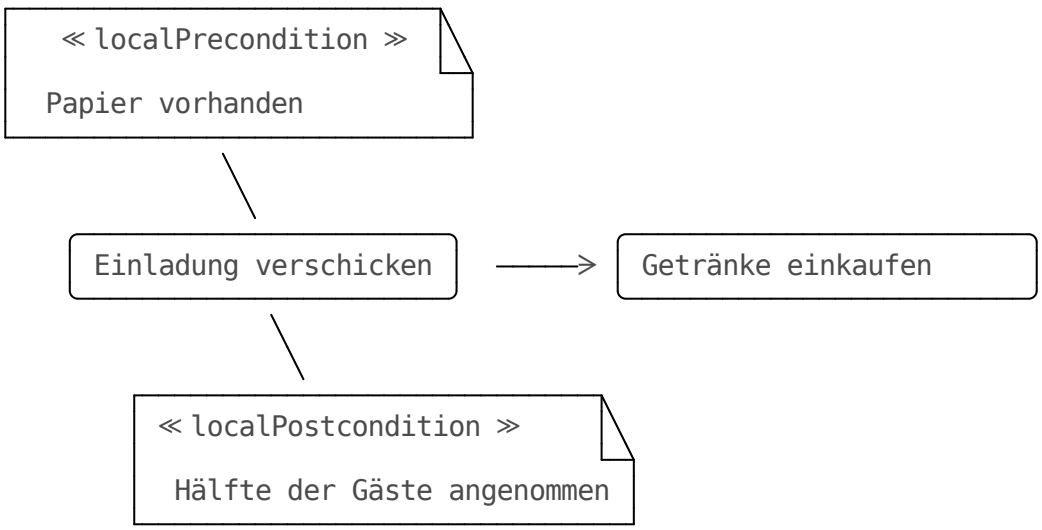
UML2 strukturiert das Konzept der Aktivitätsmodellierung neu und führt als übergeordnete Gliederungsebene Aktivitäten ein, die Aktionen, Objektknoten sowie Kontrollelemente der Ablaufsteuerung und verbindende Kanten umfasst. Die Grundidee ist dabei, dass neben dem Kontrollfluss auch der

Objektfluss modelliert wird.

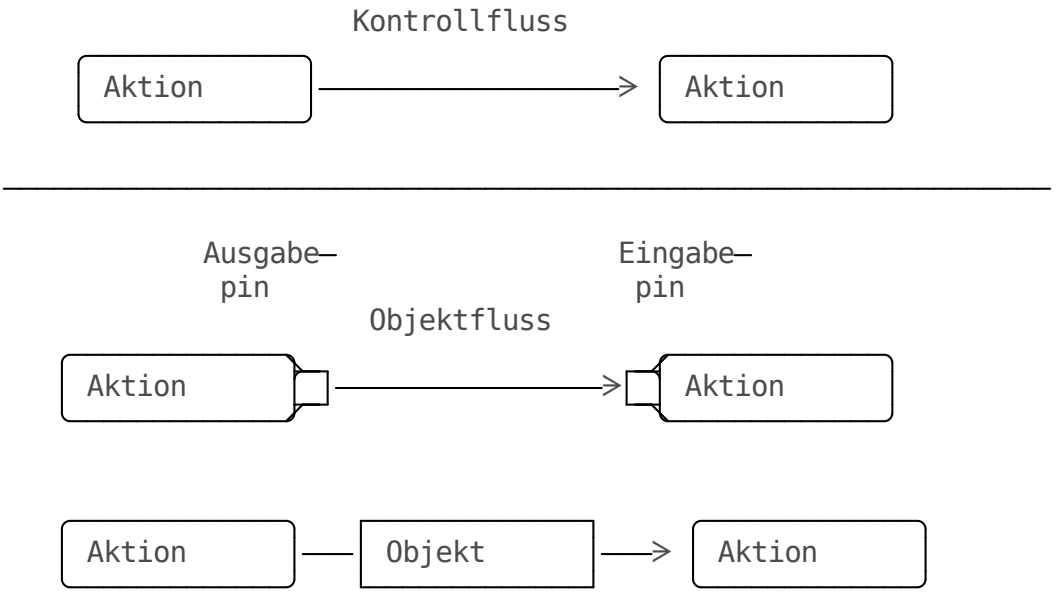
- Aktivitäten definieren Strukturierungselemente für Aktionen, die durch Ein- und Ausgangsparameter, Bedingungen, zugehörige Aktionen und Objekte sowie einen Bezeichner gekennzeichnet sind.



- Aktionen stehen für den Aufruf eines Verhaltens oder die Bearbeitung von Daten, die innerhalb einer Aktivität nicht weiter zerlegt wird.

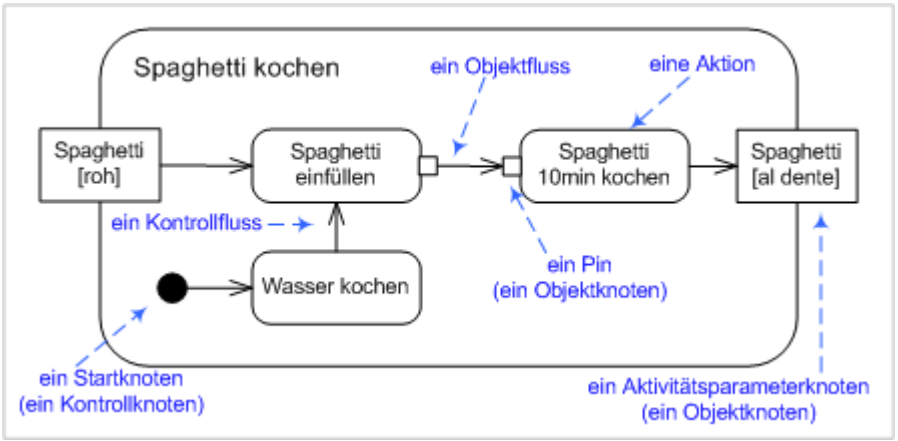


- Objekte repräsentieren Daten und Werte, die innerhalb der Aktivität manipuliert werden. Damit ergibt sich ein nebeneinander von Kontroll- und Objektfluss.



- Signale und Ereignisse sind die Schnittstellen für das Auslösen einer Aktion

Beispiel

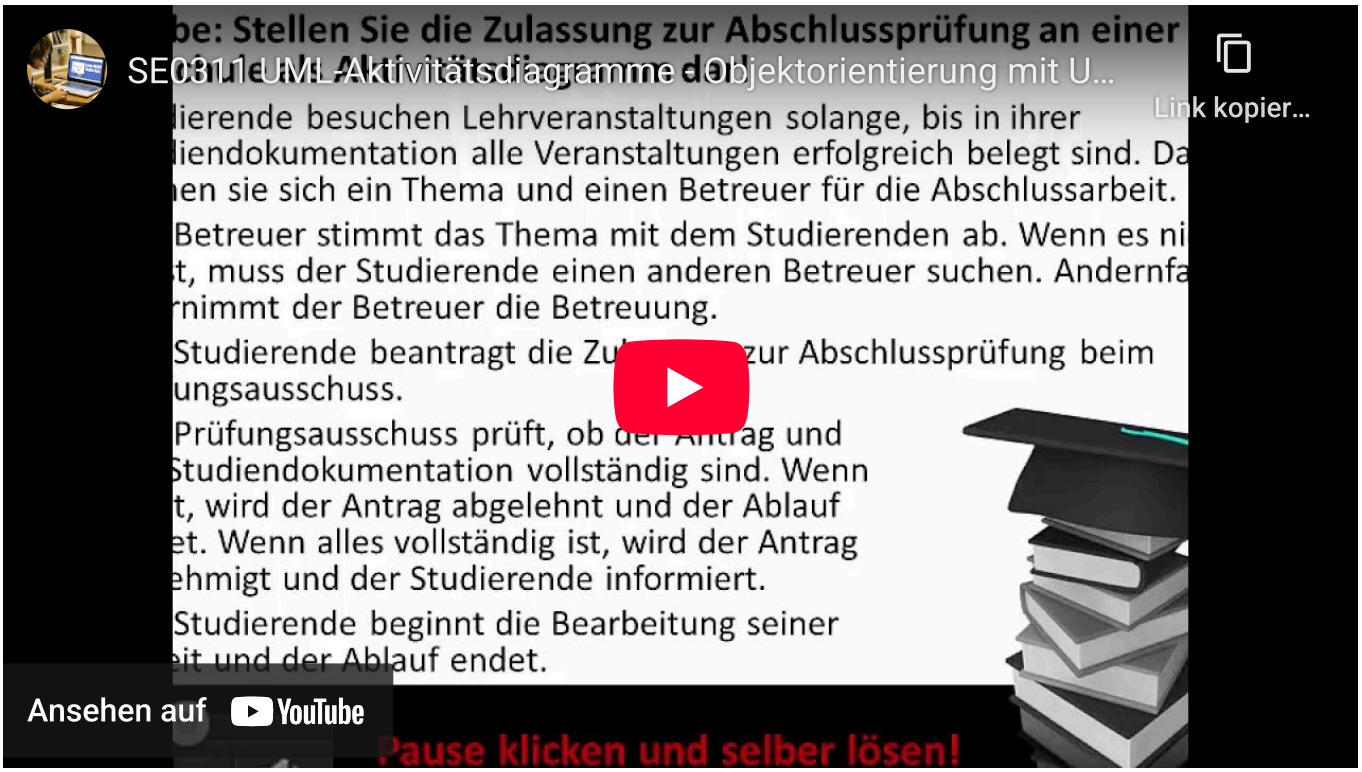


Beispiel eines Aktivitätsdiagramms - Wikimedia, Autor Gubaer, UML2 Aktivitätsdiagramm, https://commons.wikimedia.org/wiki/File:Uml-Activity-Beispiel2.svg

Anwendung

- Verfeinerung von Anwendungsfällen (aus den Use Case Diagrammen)
- Darstellung von Abläufen mit fachlichen Ausführungsbedingungen
- Darstellung für Aktionen im Fehlerfall oder Ausnahmesituationen

Das nachfolgende Video zeigt die Erstellung von Aktivitätsdiagrammen und arbeitet dabei die Unterschiede von UML 1.x und UML 2.x heraus.



Sequenzdiagramm

Sequenzdiagramme beschreiben den Austausch von Nachrichten zwischen Objekten mittels Lebenslinien.

Ein Sequenzdiagramms besteht aus einem Kopf- und einem Inhaltsbereich. Von jedem Kommunikationspartner geht eine **Lebenslinie** (gestrichelt) aus.

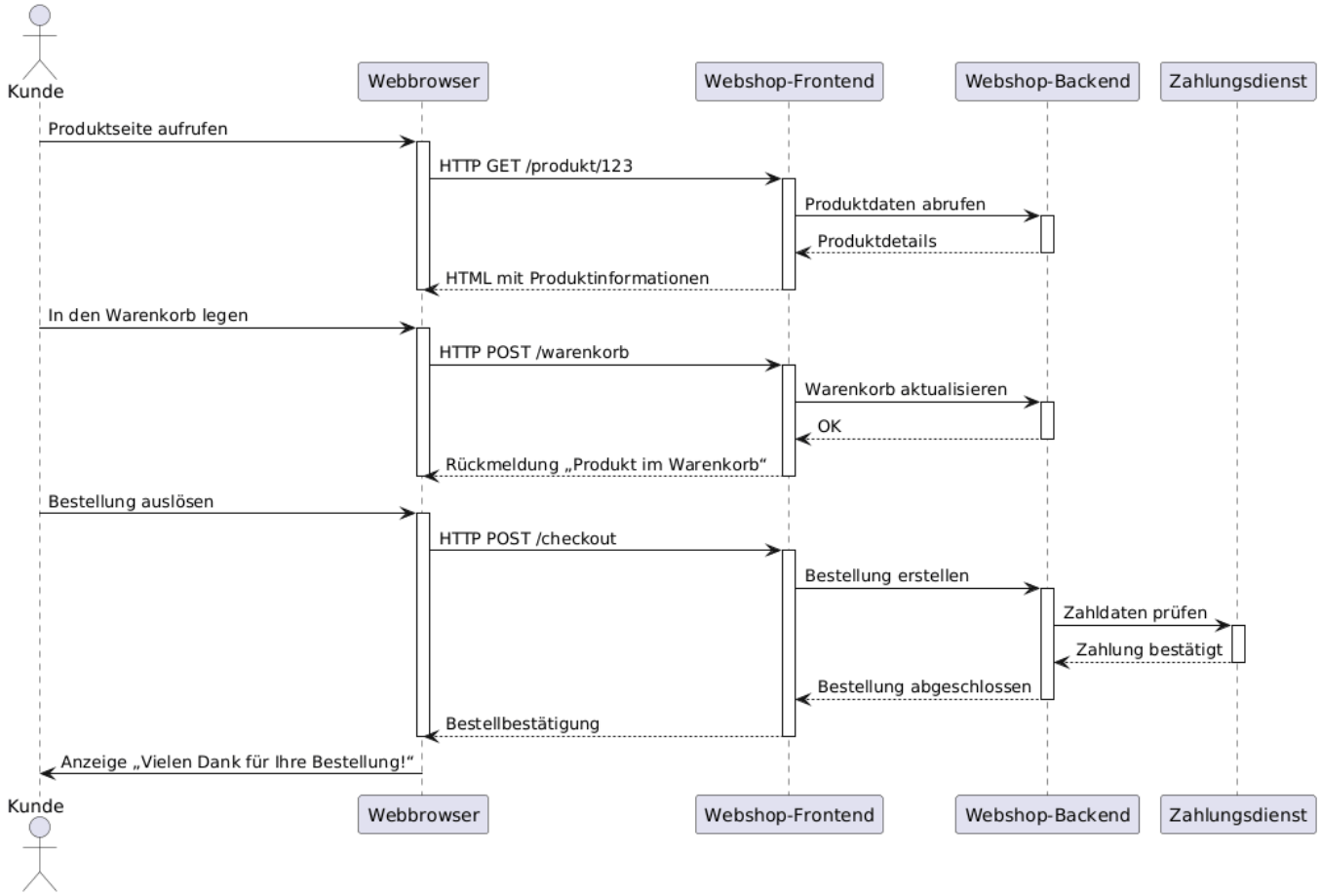
Eine Nachricht wird in einem Sequenzdiagramm durch einen Pfeil dargestellt, wobei der Name der Nachricht über den Pfeil geschrieben wird. Nachrichten können sein:

- Operationsaufrufe einer Klasse
- Ergebnisse einer Operation, Antwortsnachricht (gestrichelte Linien)
- Signale
- Interaktionen mit den Nutzern
- das Setzen einer Variablen

Für synchrone und asynchrone Operationsaufrufe (Nachrichten) gibt es verschiedene Notationsvarianten. **Synchrone Nachrichten** werden mit einer gefüllten Pfeilspitze, **asynchrone Nachrichten** mit einer offenen Pfeilspitze gezeichnet.

Die schmalen Rechtecke, die auf den Lebenslinien liegen, sind **Aktivierungsbalken**, die den Focus of Control anzeigen, also jenen Bereich, in dem ein Objekt über den Kontrollfluss verfügt, und aktiv an Interaktionen beteiligt ist.

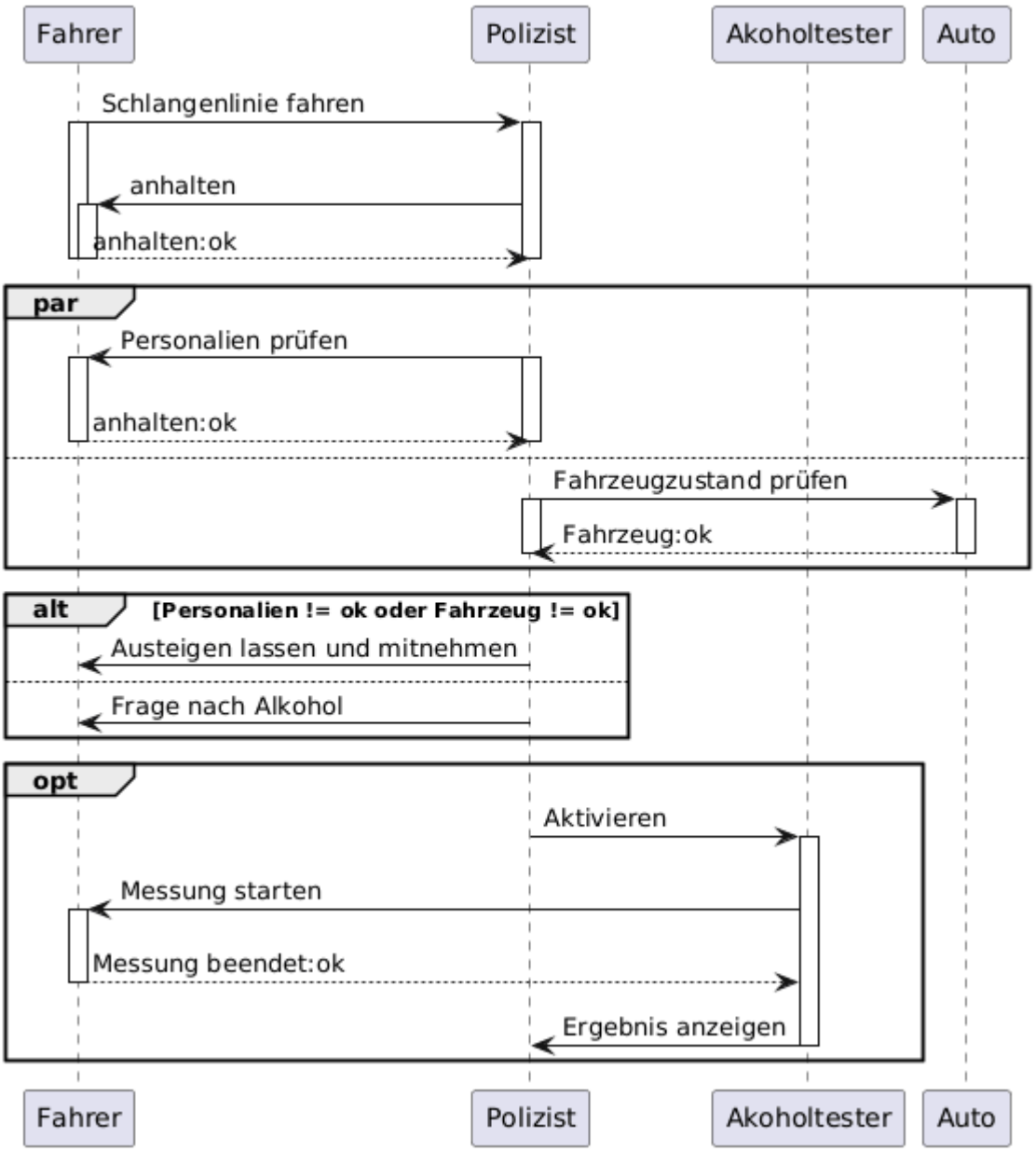
Beispiel



Bestandteile

Name	Beschreibung
Objekt	Dient zur Darstellung eines Objekts (:Klasse) im Kopfbereich.
Nachrichtensequenzen	Modelliert den Informationsfluss zwischen den Objekten
Aktivitätsbalken	Repräsentiert die Zeit, die ein Objekt zum Abschließen einer Aufgabe benötigt.
Akteur	Repräsentiert einen Nutzer (Strichmännchen)
Lebenslinien-Symbol	Stellt durch die Ausdehnung nach unten den Zeitverlauf dar.
Fragmente	Kapseln Sequenzen in Szenarien, wie optionalen Interaktionen (opt), Alternativen (alt), Schleifen (Loop), Parallelität (par)
Interaktionsreferenzen	Binden Submodelle und deren Ergebnisse ein (ref)

Beispiel



Alkoholkontrolle.plantUML

Anwendung

- Verfeinerung von Anwendungsfällen (aus den Use Case Diagrammen)
- Darstellung von Kommunikation zwischen Systemkomponenten
- Darstellung der Steuerung des Programmflusses und die
- Darstellung der Behandlung von Ausnahmen

Sequenzdiagramm vs. Aktivitätsdiagramm

- Beide können zur Beschreibung von Use Cases und einzelnen Methoden einer Klasse verwendet werden
- bieten unterschiedliche Perspektiven und Betrachtungsweisen auf diese Elemente:
- Aktivität: Visualisierung der Ablaufschritte, Darstellung/Dokumentation von komplexen Geschäftsprozessen oder Systemabläufen
- Sequenz: Darstellung von Interaktionen zwischen Objekten, einschließlich Aufrufen von Methoden einer Klasse, betonen die zeitliche Abfolge der Kommunikation

Klassendiagramme

> Ein Klassendiagramm ist eine grafischen Darstellung (Modellierung) von Klassen, Schnittstellen sowie deren Beziehungen.

Beispiel

Nehmen wir an, sie planen die Software für ein Online-Handel System. Es soll sowohl verschieden Nutzertypen (*Customer* und *Administrator*) als auch die Objekt *ShoppingCart* und *Order* umfassen.

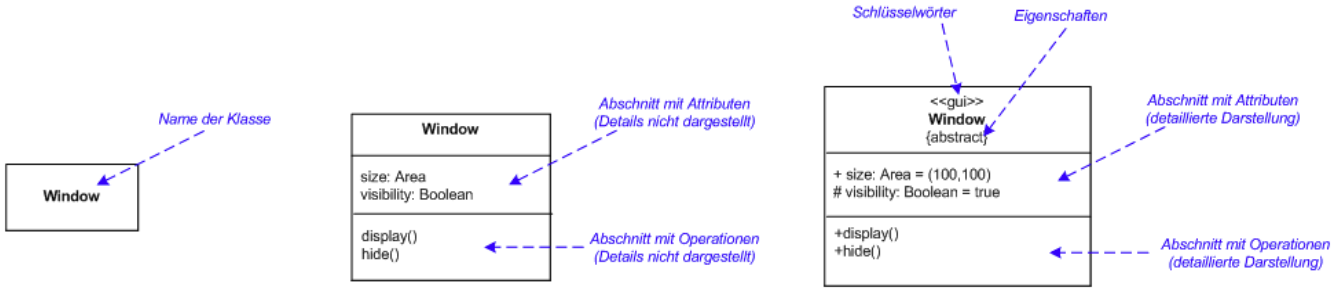
Klassen

Klassen werden durch Rechtecke dargestellt, die entweder nur den Namen der Klasse (fett gedruckt, abstrakte und Interfaces evtl. kursiv) tragen oder zusätzlich auch Attribute, Operationen und Eigenschaften spezifiziert haben. Oberhalb des Klassennamens können Schlüsselwörter in *Guillemets* ('<< >>') und unterhalb des Klassennamens in geschweiften Klammern zusätzliche Eigenschaften (wie {abstrakt}) stehen. Mit Schlüsselwörtern können zusätzliche Informationen oder Meta-Eigenschaften zur Standardsemantik der Elemente hinzugefügt werden. Sie bieten eine Möglichkeit, benutzerdefinierte Modellierungskonzepte hinzuzufügen oder vorhandene Konzepte zu präzisieren.

Elemente der Darstellung :

Eigenschaften	Bedeutung
Attribute	Beschreiben die Struktur der Objekte: die Bestandteile und darin enthaltene Daten
Operationen	Beschreiben das Verhalten der Objekte (Methoden)
Zusicherungen	Bedingungen, Voraussetzungen und Regeln, die die Objekte erfüllen müssen
Beziehungen	Beziehungen einer Klasse zu anderen Klassen

Wenn die Klasse keine Eigenschaften oder Operationen besitzt, können die entspechenden Abschnitte wegfallen.



UML Klassendiagramme und deren Attribute - adaptiert aus [WikiUMLClass]

Objekte vs. Klassen

Klassendiagramm	Beispielhaftes Objektdiagramm
	

Gegenüberstellung motiviert nach What is Object Diagram?*, https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-object-diagram/, Autor unbekannt*

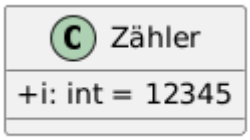
Objektdiagramm stellt die Instanzen von Klassen und ihre Beziehungen zueinander zu einem bestimmten Zeitpunkt dar basierend auf einem Klassendiagramm (ein „Schnappschuss“ der Objekte innerhalb eines Systems) und enthält als Elemente: Objekte (Instanzen der Klasse), Links (Instanzen von Assoziationen), Attribute (Werte von Eigenschaften von Objekten).

Anwendung des Objektdiagramms:

- Um den Zustand eines Systems zu einem bestimmten Zeitpunkt darzustellen bzw. während der Laufzeit zu visualisieren.
- Um Testfälle und Szenarien zu beschreiben.

Merke: Vermeiden Sie bei der Benennung von Klassen, Attributen, Operationen usw. sprachspezifische Zeichen

Modellierung in UML



Ausführbarer Code in Python 2

KlasseMitUmlaut.py

```
1  class Zaehler:
2      """A simple example class"""
3      i = 12345
4
5  A = Zaehler()
6  print A.i
```

```
Listing . ...
Compiling ./main.py ...

Listing . ...
Compiling ./main.py ...

12345
12345
```

Ausführbarer Code in C++ 20

KlasseMitUmlaut.cpp

```cpp
#include <iostream>

class Zähler{
  public:
   int i = 12345;
};

int main()
{
  Zähler A = Zähler();
  std::cout << A.i;
  return 0;
}
```

```
1234512345
```

Sichtbarkeitsattribute

Zugriffsmodifizierer	Innerhalb eines Assemblys		Außerhalb eines Assemblys	
	Vererbung	Instanzierung	Vererbung	Instanzierung
`public`	ja	ja	ja	ja
`private`	nein	nein	nein	nein
`protected`	ja	nein	ja	nein
`internal`	ja	ja	nein	nein
`ìnternal protected`	ja	ja	ja	nein

public, private

Die Sichtbarkeitsattribute `public` und `private` sind unabhängig vom Vererbungs-, Instanzierungs- oder Paketstatus einer Klasse. Im Beispiel kannn der TrafficOperator nicht auf die Geschwindigkeiten der Instanzen von Car zurückgreifen.

protected

Die abgeleitete Klassen Bus und PassagerCar erben von Car und übernehmen damit deren Methoden. Die Zahl der Sitze wird beispielsweise mit ihrem Initialisierungswert von 5 auf 40 gesetzt. Zudem muss die Methode `StopAtStation` auch auf die Geschwindigkeit zurückgreifen können.

internal

Ein Member vom Typ `internal` einer Basisklasse kann von jedem Typ innerhalb seiner enthaltenden Assembly aus zugegriffen werden.

Merke: Der UML Standard kennt nur `+ public`, `- private`, `# protected` und `~ internal`. Das C# spezifische `internal protected` ist als weitere Differenzierungsmöglichlichkeit nicht vorgesehen.

Attribute

Merke: In der C# Welt sprechen wir bei Attributen von Membervariablen und Feldern.

Im einfachsten Fall wird ein Attribut durch einen Namen repräsentiert, der in der Klasse eindeutig sein muss - Die Klasse bildet damit den Namensraum der enthaltenen Attribute.

Entsprechend der Spezifikation sind folgende Elemente eines Attributes definierbar:

```
[Sichtbarkeit] [/] Name [: Typ] [ Multiplizität ] [= Vorgabewert]
  [{Eigenschaftswert}]
```

- *Sichtbarkeit* ... vgl. vorheriger Absatz Das "/" bedeutet, dass es sich um ein abgeleitetes Attribut handelt.
- *Name* ... des Attributes, Leer und Sonderzeichen sollten weggelassen werden, um zu vermeiden, dass Sie bei der Implementierung Probleme generieren.
- *Typ* ... UML verwendet zwar einige vordefinierte Typen (Integer, Real, String, Boolean, UnlimitedNatural) beinhaltet aber keine Einschränkungen zu deren Wertebereich!
- *Multiplizität* ... die Zahlenwerte in der rechteckigen Klammer legen eine Ober- und Untergrenze der Anzahl (Kardinalitäten) von Instanzen eines Datentyps fest.

Beispiel	Bedeutung
`0..1`	optionales Attribut, das aber höchstens in einer Instanz zugeordnet wird
`1..1`	zwingendes Attribut
`0..n`	optionales Attribute mit beliebiger Anzahl
`1..*`	zwingend mit beliebiger Anzahl größer Null
`n..m`	allgemein beschränkte Anzahl größer 0

- *Vorgabewerte* ... definieren die automatische Festlegung des Attributes auf einen bestimmten Wert
- *Eigenschaftswerte* ... bestimmen die besondere Charakteristik des Attributes

Eigenschaft	Bedeutung
`readOnly`	unveränderlicher Wert
`subsets`	definiert die zugelassene Belegung als Untermenge eines anderen Attributs
`redefines`	überschreiben eines ererbten Attributes
`ordered`	Inhalte eines Attributes treten in geordneter Reihenfolge ohne Duplikate auf
`bag`	Attribute dürfen ungeordnet und mit Duplikaten versehen enthalten sein
`sequence`	legt fest, dass der Inhalt sortiert ist, Duplikate sind erlaubt
`composite`	starke Abhängigkeitsbeziehungen

Daraus ergeben sich UML-korrekte Darstellungen

Attributdeklaration	Korrekt	Bemerkung
`public zähler:int`	ja	Umlaute sind nicht verboten
`/ alter`	ja	Datentypen müssen nicht zwingend angegeben werden
`privat adressen: String [1..*]`	ja	Menge der Zeichenketten
`protected bruder: Person`	ja	Datentyp kann neben den Basistypen jede andere Klasse oder eine Schnittstelle sein
String	nein	Name des Attributes fehlt
privat, public name: String	nein	Fehler wegen mehrfachen Zugriffsattributen


Protected

Korrespondierendes UML-Klassendiagramm AttributeExample.plantUML

AttributeExample

```
using System;

class Example
{
  int attribute1;
  public int attribute2;
  public static double pi = 3.14;
  private bool attribute3;
  protected short attribute4;
  internal const string attribute5 = "Test";
  B attribute6;
  System.Collections.Speciallized.StringCollection attribute7;
  Object attribute8{
    get{return wert * 10;}
  }
}
```

Operationen

> **Merke:** In der C# Welt sprechen wir bei Operationen von Methoden.

Operationen werden durch mindestens ihren Nahmen sowie wahlfrei weitere Angaben definiert. Folgende Aspekte können entsprechend der UML Spezifikation beschrieben werden:

```
[Sichtbarkeit] Name (Parameterliste) [: Rückgabetyp] [{Eigenschaftswert}]
```

Dabei ist die Parameterliste durch folgende Elemente definiert:

```
[Übergaberichtung] Name [: Typ] [Multiplizität] [=
  Vorgabewert][{Eigenschaftswert}]
```

- *Sichbarkeit* ... analog Attribute
- *Name* ... analog Attribute
- *Parameterliste* ... Aufzählung der durch die aufrufende Methode übergebenden Parameter, die im folgenden nicht benannten Elemente folgend den Vorgaben, die bereits für die Attribute erfasst wurden:
  - *Übergaberichtung* ... Spezifiziert die Form des Zugriffes (`in` = nur lesender Zugriff, `out` = nur schreibend (reiner Rückgabewert), `inout` = lesender und schreibender Zugriff)
  - *Vorgabewert* ... default-Wert einer Übergabevariablen
- *Rückgabetyp* ... Datentyp oder Klasse, der nach der Operationsausführung zurückgegeben wird.
- *Eigenschaftswert* ... Angaben zu besonderen Charakteristika der Operation


Protected

Korrespondierendes UML-Klassendiagramm

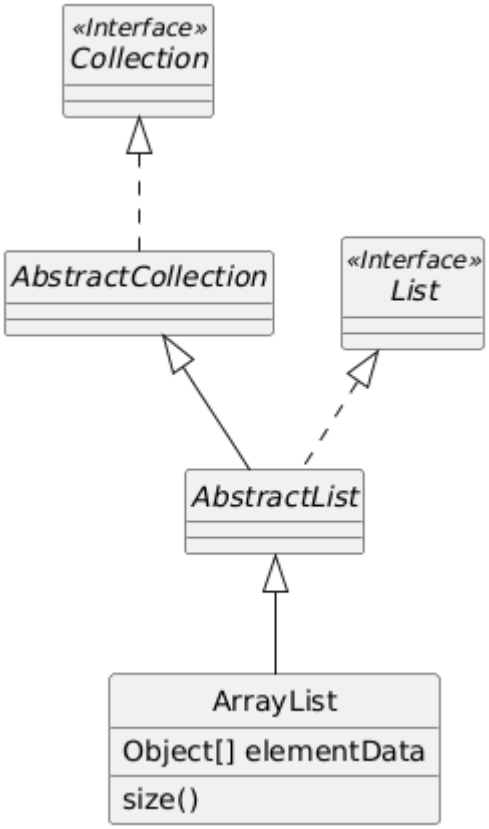
OperationsExample

```
using System;

class Example
{
  public static void operation1(){
    // Implementierung
  }

  private int operation2 (int param1 = 5)
  {
    // Implementierung
    return value;
  }

  protected void operation3 (ref C param2)
  {
    // Implementierung
    param3 = ...
  }

  internal B operation4 (out StringCollection param3)
  {
    // Implementierung
    return value;
  }
}
```

[WikiUMLClass] https://de.wikipedia.org/wiki/Klassendiagramm#/media/Datei:UmlCd_Klasse-3.svg, Autor Gubaer, CC BY-SA 3.0

Schnittstellen

Eine Schnittstelle wird ähnlich wie eine Klasse mit einem Rechteck dargestellt, zur Unterscheidung aber mit dem Schlüsselwort `interface` gekennzeichnet.



Darstellung von Schnittstellen in UML-Klassendiagrammen

Eine alternative Darstellung erfolgt in der LolliPop Notation, die die grafische Darstellung etwas entkoppelt.

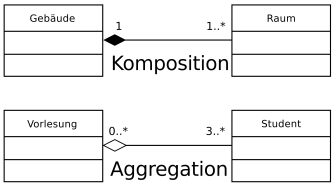
Steigerung der Lesbarkeit durch die Verwendung von Lollipop Symbolen lollipop.plantUML

OperationsExample

```
using System;

interface Sortierliste{
  void einfuegen (Eintrag e);
  void loeschen (Eintrag e);
}

class Datenbank : SortierteListe
{
  void einfuegen (Eintrag e) {//Implementierung};
  void loeschen (Eintrag e) {//Implementierung};
}
```

Beziehungen

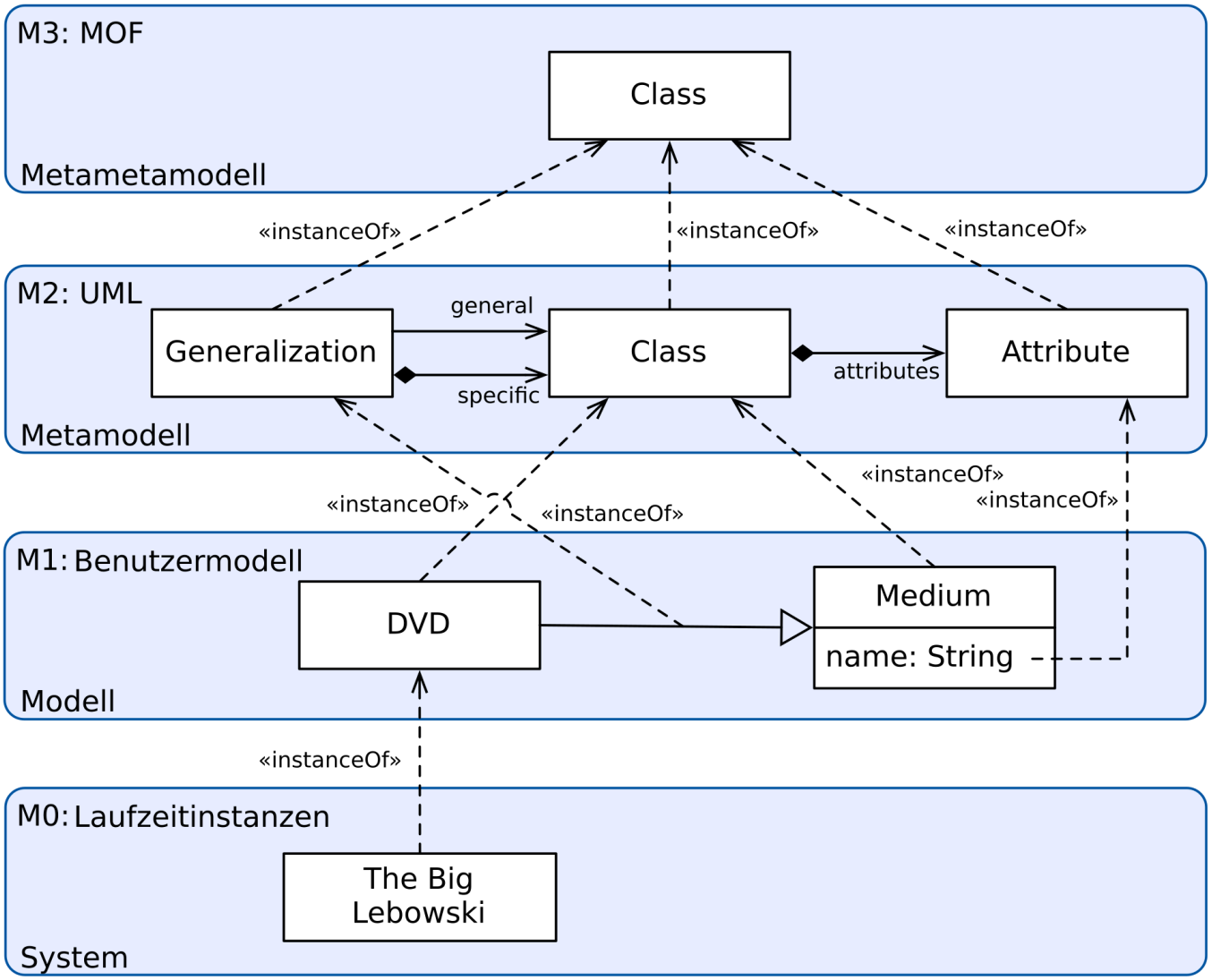
Die Möglichkeiten der Verknüpfung zwischen Klassen und Interfaces lassen sich wie folgt gliedern:

Beziehung	Darstellung	Bedeutung
Generalisierung		gerichtete Beziehung zwischen einer generelleren und einer spezielleren Klasse (Vererbung)
Assoziationen (ohne Anpassung)		beschreiben die Verknüpfung allgemein
Assoziation (Komposition/Aggregation)		Bildet Beziehungen von einem Ganzen und seinen Teilen ab

UML- Metamodell

Abstraktionseben der UML-Modellierung:

- M0 - Instanzebene: repräsentiert die konkrete Ausführung des Systems und ist nicht direkt in der UML-Modellierung abgebildet, auf dieser Ebene befinden sich die tatsächlichen Instanzen von Objekten, die während der Laufzeit eines Systems existieren.
- M1 - Modellierungsebene: beinhaltet verschiedene Arten von UML-Diagrammen wie Klassendiagramme, Aktivitätsdiagramme, Zustandsdiagramme usw. für eigentliche Benutzermodelle.
- M2 - Metamodell-Ebene: beinhaltet die Modelle, die das System selbst beschreiben. Das UML-Metamodell ist ein Beispiel für ein Metamodell auf dieser Ebene. Es beschreibt die Struktur und Syntax der UML selbst und ermöglicht es, UML-Diagramme zu erstellen und zu interpretieren.
- M3 - Metametamodell-Ebene: Diese Ebene beschreibt die Struktur und Semantik von Metamodellen auf der M2-Ebene. Metametamodelle definieren die Regeln und Konzepte, die verwendet werden, um Metamodelle zu erstellen.

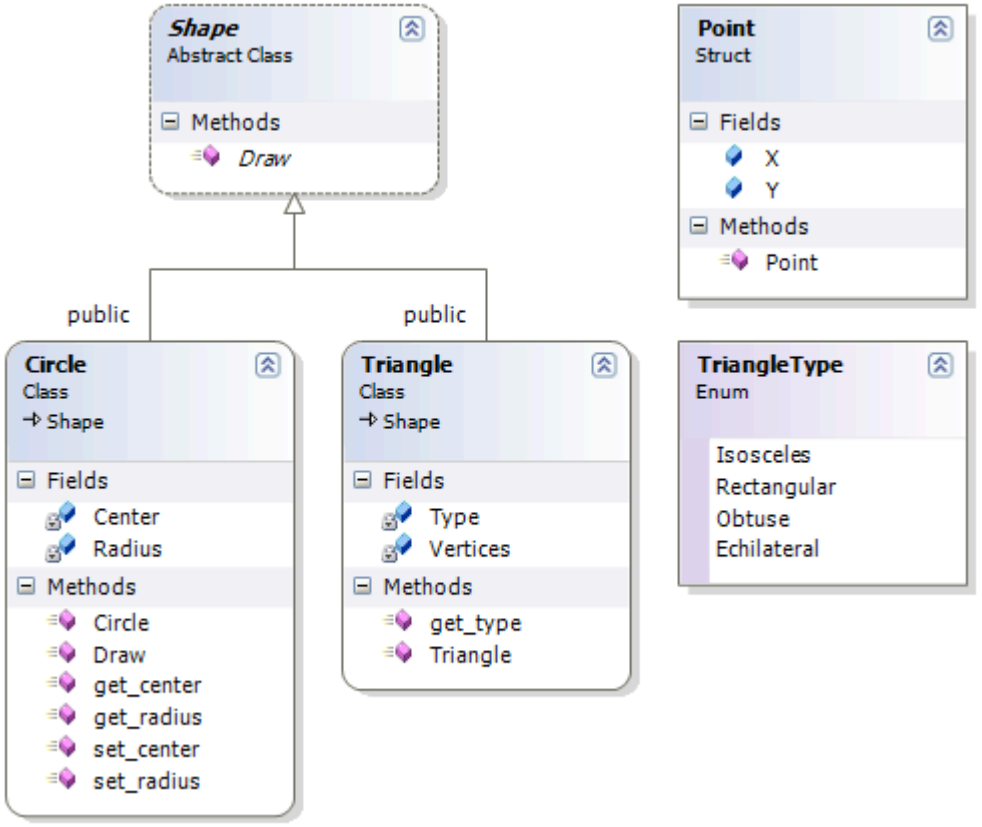


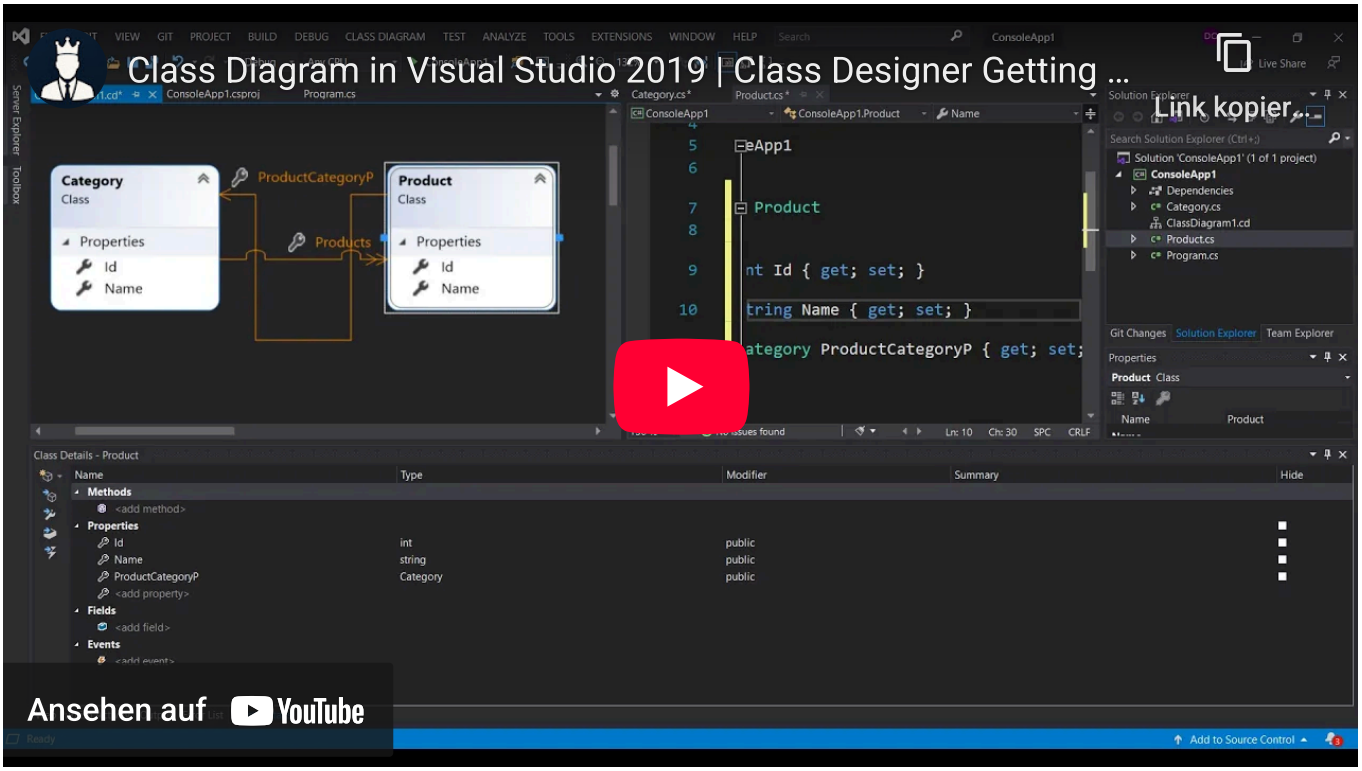
Von Jens von Pilgrim - created by author, based on OMG: Unified Modeling Language: Infrastructure. Page 31, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=9914721

Verwendung von UML Tools

Verwendung von Klassendiagrammen

- ... unter Umbrello (UML Diagramm Generierung / Code Generierung)
- ... unter Microsoft Studio Link)





- ... unter Visual Studio Code mit PlantUML oder UMLet (siehe Codebeispiele zu dieser Vorlesung)
- ... unter Visual Studio Code mit PlantUmlClassDiagramGenerator

Aufgaben

- [ ] Experimentieren Sie mit Umbrello
- [ ] Experimentieren Sie mit der automatischen Extraktion von UML Diagrammen für Ihre Computer-Simulation aus den Übungen
- [ ] Evaluieren Sie das Add-On "Class Designer" für die Visual Studio Umgebung