Objektorientierte Systemanalyse

Teil 1: Einführung

IT works.

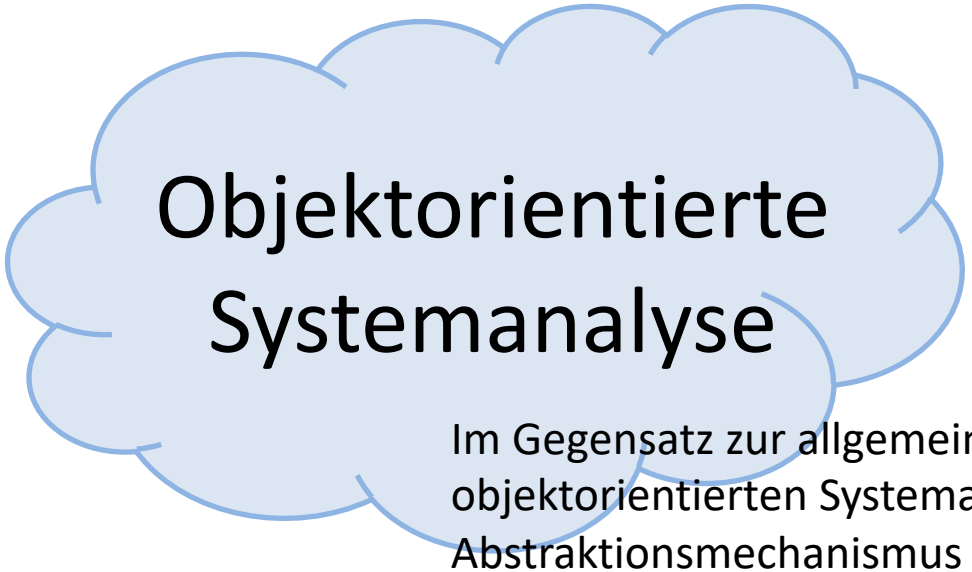
Zu mir, Klaus Mairon, ...

- Mein Arbeitgeber
  - msg nexinsure ag, Unternehmen der msg-Gruppe mit dem Schwerpunkt auf Versicherungssoftware
  - Geschäftsstelle St. Georgen, Konzernzentrale in Ismaning/München
- Meine Aufgaben bei der msg nexinsure ag
  - Principal IT-Consultant IT-Architecture
  - Berater und Softwarearchitekt, Abteilungsleiter Architektur
- Mein Hintergrund
  - Studium der Wirtschaftsinformatik (Diplom) und Application Architectures (Master) an der Hochschule Furtwangen
  - Promotion an der University of Plymouth, United Kingdom
  - Dozententätigkeit an der Hochschule Furtwangen 1997 bis 2010
  - Dozententätigkeit an der Dualen Hochschule Baden-Württemberg, Villingen-Schwenningen seit 2009
  - Mitglied diverser Prüfungskommisionen im Bereich Wirtschaftsinformatik der DHBW Villingen-Schwenningen seit 2004
  - Ortschaftsrat in VS-Weigheim seit 2009
  - Leiter der Jugendabteilung des FC „Vorwärts" 1920 Weigheim e.V.

Agenda

- **Einführung**
  - Basiskonzepte, UML
  - Fokus in der OOA
- Use Cases
  - Use Case Diagramme
  - Textschablone
- Aktivitätsdiagramme
  - Grundkonzepte
  - Erweiterungen
- Klassendiagramme
  - Klassendiagramme
  - Paketdiagramme, Komponentendiagramme
  - Objektmodelle, Szenarien, Zustandsautomaten
  - CRC-Karten
- UI-Design
  - Ergonomie, Usability, Gestaltung
  - Ableitung aus Klassendiagrammen
  - Beschreibung über Zustandsautomaten

Worum es geht ...

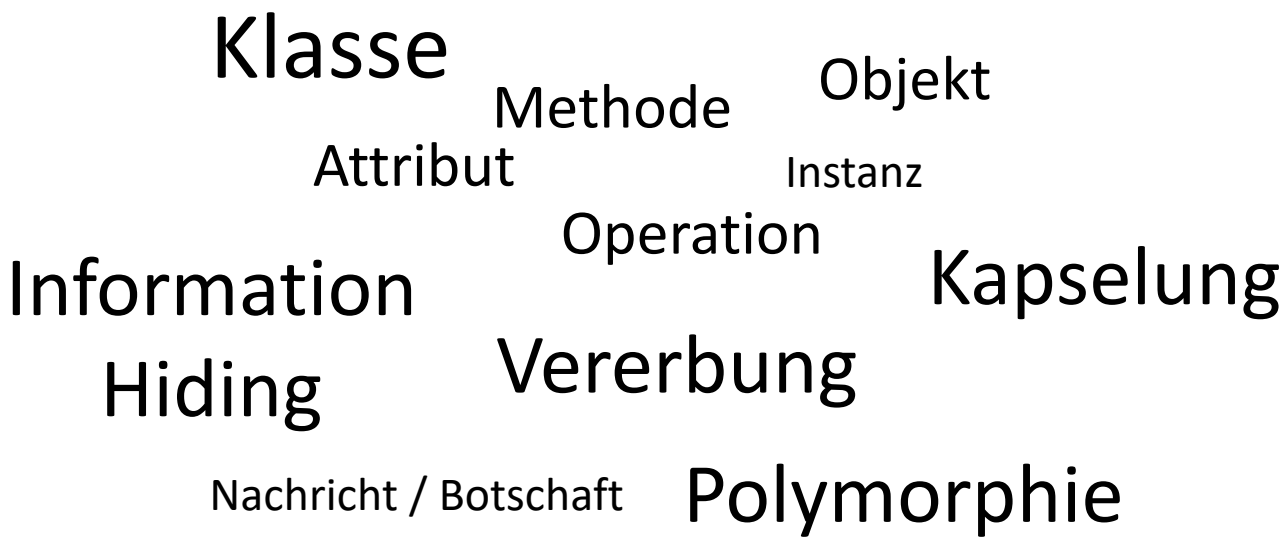


Im Gegensatz zur allgemeinen Systemanalyse wird in der objektorientierten Systemanalyse mit dem einheitlichen Abstraktionsmechanismus der Objektorientierung gearbeitet, um die immer komplexeren Anforderungen an Software-Systeme vollständig beschreiben zu können.

Durch die Objektorientierung werden durchgängige Konzepte, Modelle, Methoden und Werkzeuge über alle Phasen der Systementwicklung eingesetzt.

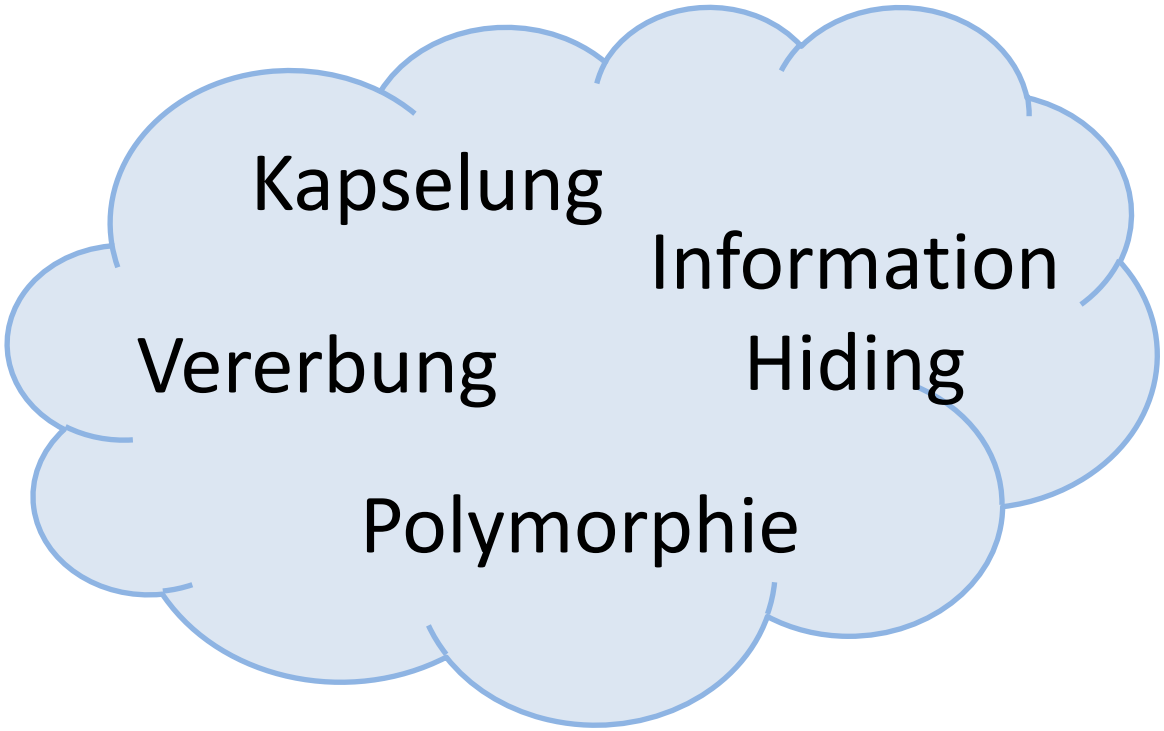
Begriffe der Objektorientierung

Im Kontext der Objektorientierung treffen wir auf eine ganze Reihe von Begriffen, die teilweise synonym verwendet werden und für das Verständnis des Konzepts notwendig sind:



Grundkonzepte der Objektorientierung

Vier dieser Begriffe sind besonders hervorzuheben: Definieren diese doch die grundsätzlichen Konzepte der Objektorientierung:

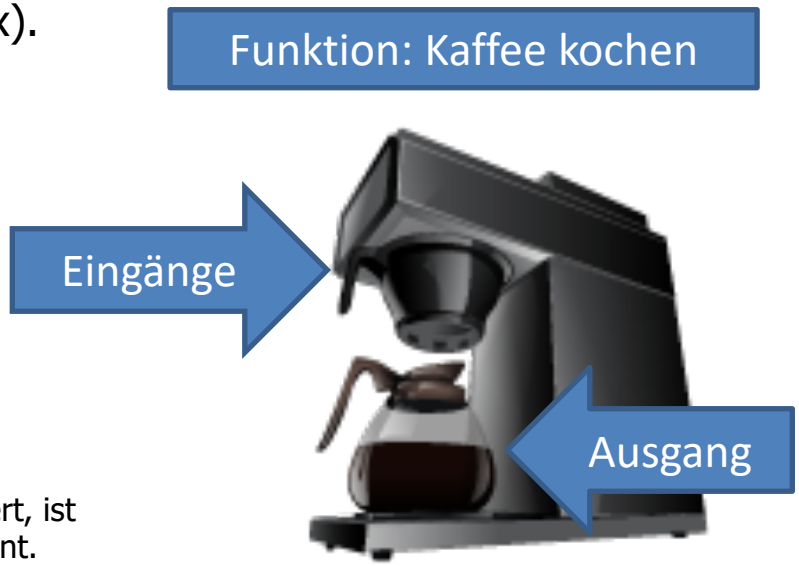


Kapselung und Information Hiding

- Abstrakter Datentyp (ADT)
  - Definition eines neuen Datentyps, welcher einerseits die Beschreibung einer Datenstruktur umfasst und andererseits die darauf zulässigen Operationen festlegt.

- Kapselung bzw. Information Hiding
  - Der Zugriff auf die Daten kann nur über die festgelegten Operationen erfolgen.
  - Die Daten sind nach Außen nicht sichtbar, sie sind innerhalb des abstrakten Datentyps gekapselt.
  - Der Zugriff auf die Daten wird durch die Operationen kontrolliert.

- Daten sind nach außen hin nicht sichtbar (Black Box).
  - Man kennt Art und Bedeutung der benötigten Eingänge.
  - Man kennt Art und Bedeutung der gelieferten Ausgänge.
  - Man kennt die Funktion.
  - Man weiß nicht, wie die Black Box ihre Aufgabe erfüllt.
  - Beispiel Kaffeemaschine:
    - Es ist bekannt, welcher Input über welche Schnittstellen erwartet wird (Wasser, Kaffeepulver).
    - Es ist bekannt, welcher Output über welche Schnittstellen geliefert wird (Ausfluss Kaffeekanne)
    - Auf welche Weise die Kaffeemaschine intern konkret funktioniert, ist nicht bekannt und für den Erhalt des Ergebnisses auch irrelevant.



Kapselung in der Objektorientierung

- Die Definition von Datenstrukturen und die Definition von Funktionen auf diese Datenstrukturen werden in einer Einheit, der sogenannten **Klasse**, zusammengefasst.

- Der Zugriff auf die Datenstruktur erfolgt nur über die definierten Funktionen. Hierdurch kann sichergestellt werden, dass die Werte der Datenstruktur nur fachlich korrekte Ausprägungen annehmen.

- Beispiel einer Klassendefinition in Python:

```
class Girokonto:
    __saldo = 0.0
    __dispo = -1000.0

    def einzahlen(self, betrag):
        self.__saldo+=betrag
        print("Neuer Saldo: " + str(self.__saldo))

    def auszahlen(self, betrag):
        if (self.__saldo - betrag > self.__dispo):
            self.__saldo-=betrag
            print("Neuer Saldo: " + str(self.__saldo))
        else:
            print("Betrag kann nicht ausgezahlt werden.")
```

Klassenname

Datenstruktur -> Attribute

Funktionen -> Methoden

Klassen und Objekte, Attribute und Operationen

- Definition **Klasse**:
  Eine Klasse definiert (wie eine Schablone) für eine beliebige Anzahl von Objekten deren Datenstruktur (Attribute), das Verhalten (Methoden) und Beziehungen zu anderen Klassen.

- Definition **Objekt**:
  - Ein Objekt ist eine konkretes Exemplar oder Instanz seiner Klasse.
  - Ein Objekt besitzt einen Zustand, der sich über die aktuellen Attributwerte und Verbindungen zu anderen Objekten definiert.
  - Ein Objekt reagiert mit einem definierten Verhalten (ausgelöst durch Methoden-Aufrufe) auf seine Umgebung.
  - Jedes Objekt besitzt eine eindeutige Objektidentität, die es von allen anderen Objekten unterscheidet. Deshalb können Objekte auch unterschieden werden, wenn ihr Zustand (ihre Attributwerte) gleich sind.

- Synonyme:
  - **Instanz** = Objekt: Bezeichnung eines Exemplars einer Klasse; Instanziieren einer Klasse bedeutet das Erzeugen eines Objekts einer Klasse.
  - **Methode** = Operation: Definition einer Funktion einer Klasse, welche auf der Datenstruktur operiert. Das Aufrufen einer Methode/Operation/Funktion wird in der Objektorientierung auch als das senden einer Nachricht oder Botschaft bezeichnet.

Vererbung

- Vererbung definiert eine neue Klasse auf Basis der bestehenden Definition einer existierenden Klasse.
- Die neue Klasse „erbt" die Attribute (Datenstruktur), das Verhalten (Methoden) und Beziehungen der Basisklasse und erweitert diese um ihre speziellen Eigenschaften.
- Die neue Klasse kann geerbtes Verhalten durch sogenanntes „Überschreiben" der Operationen neu definieren. Das bedeutet, dass die neue Klasse eine gleiche Methode definiert wie die Basisklasse, diese sich jedoch in ihrer Implementierung unterscheidet.
- **Definition Vererbung:**
  Die Vererbung beschreibt die Beziehung zwischen einer allgemeineren Klasse und einer oder mehreren spezialisierten Klasse. Die spezialisierte Klasse erweitert die Liste der Attribute, Operationen und Beziehungen der Basisklasse. Operationen der Basisklasse dürfen redefiniert (überschrieben) werden. Es entsteht eine Klassenhierarchie oder Vererbungs-struktur.
- Synonyme:
  - Oberklasse = Superklasse = Basisklasse: allgemeinere Klasse (man spricht von **Generalisierung**)
  - Unterklasse = Subklasse: spezialisierte Klasse (man spricht von **Spezialisierung**)

Polymorphie

- **Polymorphie** ist eines der wirkungsvollsten Konzepte der Objektorientierung. Die Polymorphie bedeutet, dass das tatsächliche Verhalten eines Objekts vom konkreten, aktuellen Typ eines Objekts abhängt und nicht vom deklarierten Typ.

- **Definition Polymorphie:**
  Ein Name (Bezeichner, Variablenname) kann Objekte verschiedener Klassen bezeichnen. Jedes Objekt, das durch diesen Namen bezeichnet wird, kann auf die gleiche Botschaft (Aufruf einer Methode) auf seine eigene Art und Weise reagieren. Polymorphie bedingt in einer objektorientierten Programmiersprache den Mechanismus des späten bzw. dynamischen Bindens.

- **Definition polymorphe Operation:**
  Eine polymorphe Operation ist eine Operation, die erst zur Ausführungszeit an ein bestimmtes Objekt gebunden wird. Man spricht von spätem Binden (late binding) bzw. vom dynamischen Binden (dynamic binding).

Polymorphie – ein Beispiel

- Gegeben sei der allgemeine Begriff eines Gefäßes
- Als Verhalten sei definiert: Gefäß füllen, Gefäß leeren

- Aussage zur Polymorphie: Das tatsächliche Verhalten hängt vom konkreten, aktuellen Typ eines Objekts ab und nicht vom deklarierten Typ.

  - Deklarierter Typ der Objekte: alle Objekte gehören zur Klasse Gefäß
  - Konkret gehören die Objekte aber zu folgenden Subklassen: Glas, Gießkanne, Bierfass
  - Wie ist das konkrete Verhalten der Objekte?



Polymorphie – ein Beispiel

- Einfache Umsetzung in Python zum Ausprobieren ...

```
class Gefaess():
  def fuellen(self): pass
  def leeren(self): pass

class Giesskanne(Gefaess):
  def fuellen(self):
    print("... eintauchen")
  def leeren(self):
    print("... gießen")

class Glas(Gefaess):
  def fuellen(self):
    print("... einschenken")
  def leeren(self):
    print("... trinken")

class Fass(Gefaess):
  def fuellen(self):
    print("... einkaufen")
  def leeren(self):
    print("... zapfen")

if __name__ == "__main__":
  gefaesse = [Fass(), Glas(), Glas(), Giesskanne()]
  for g in gefaesse:
    g.fuellen()
    g.leeren()
```

Definition der Oberklasse `Gefaess`

Definition der Subklassen `Giesskanne`, `Glas` und `Fass`. Jeweils mit einem anderen Verhalten für die Methoden `fuellen()` und `leeren()`.

Definition einer Liste `gefaesse` mit unterschiedlichen Objekten der Subklassen. In einer Schleife wird von jedem Objekt die Methode `leeren()` aufgerufen.
Beachte: Es muss nicht (z.B. über eine if-Abfrage) unterschieden werden, von welchem konkreten Typ das Objekt ist. Trotzdem wird die zum konkreten Typ (z.B. `Glas`) gehörige Methode aufgerufen.

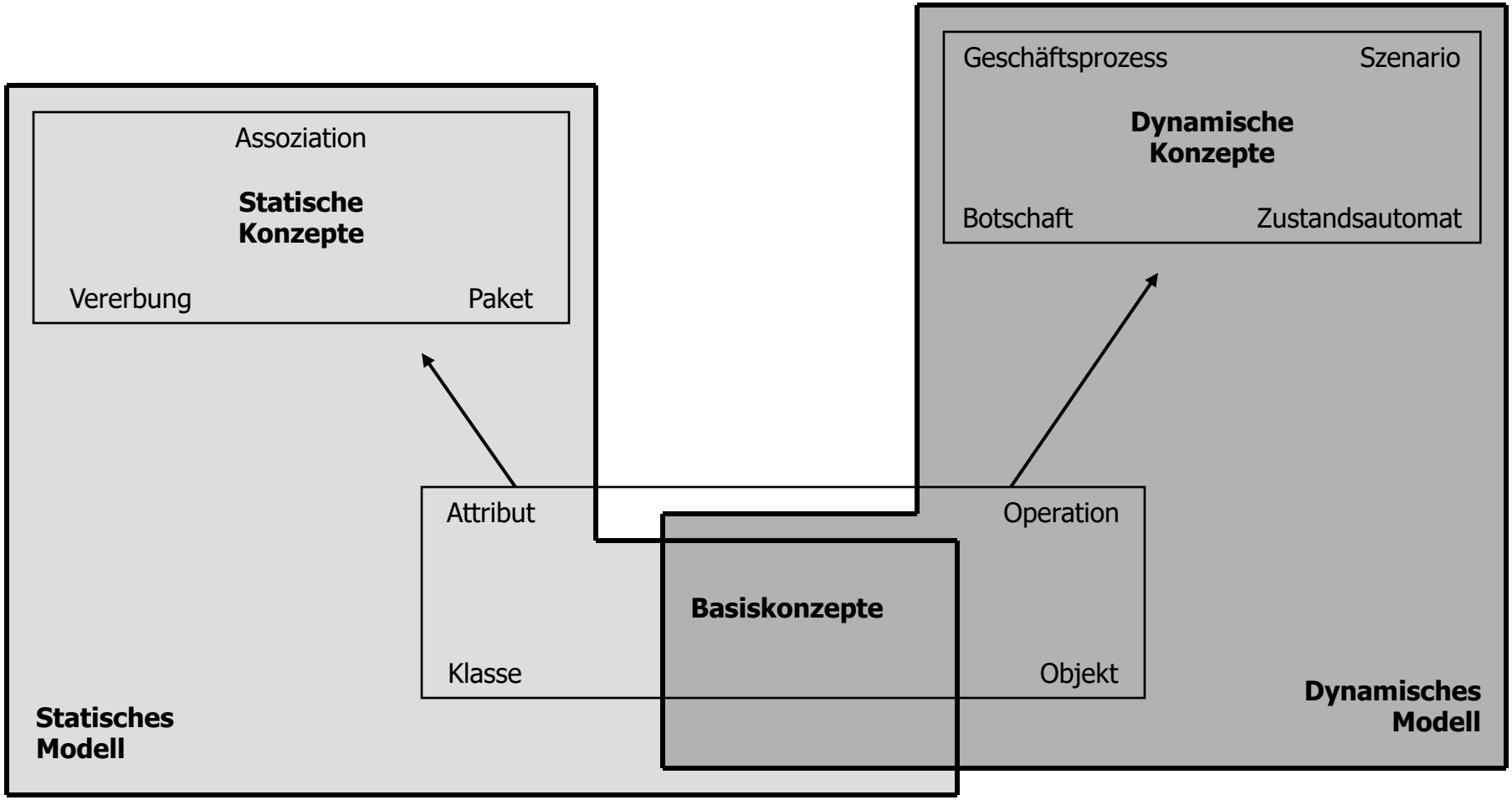
Modelle in der Software-Entwicklung

- Modelle oder Modell-Zeichnungen sind aus vielen Anwendungsbereichen bekannt.
  Zum Beispiel:
  - Grundriss oder Modell eines Hauses bzw. Architektur-Skizze
  - Schaltplan eines elektrischen Schaltkreises
  - Prototypisches Modell eines Fahrzeugs

- Modelle in der Softwareentwicklung dienen der Beschreibung eines bestehenden oder zu entwickelnden Systems. Sie helfen dabei, die Komplexität eines Systems mittels Abstraktion auf die wesentlichen Elemente begreifbar zu machen.
  Sie beschreiben zum Beispiel:
  - die benötigten Datenstrukturen und deren Zusammenhänge (z.B.: ein Kunde hat mehrere Lieferadressen)
  - mögliche Systemzustände und deren gültigen Zustandsübergänge (z.B. eine Maschine darf nur gestartet werden, wenn die Kühlung aktiviert ist)
  - das Verhalten des Systems bei Eintritt von bestimmten Ereignissen (z.B. bei Buchung der Vorauskasse wird die Ware versandt).

- Die Modellierung von Softwaresystemen kann sowohl die Anforderungen an ein System umfassen, als auch die technische Umsetzung von Anforderungen beschreiben. Die Modelle konzentrieren sich dabei jedoch immer auf zwei Aspekte:
  - Die Datenstrukturen und ihre Zusammenhänge: modelliert im statischen Modell
  - Das Verhalten eines Systems: modelliert im dynamischen Modell

Statisches und dynamisches Modell und die objektorientierten Konzepte



Quelle: H.Balzert

Was ist die UML?

- Eine standardisierte Sprache zur Spezifikation, Konstruktion, Visualisierung und Dokumentation von Modellen für Softwaresysteme.
- Die UML-Spezifikation umfasst hierzu die Definition von Semantik und grafischer Notation von Modellelementen.
- Standardisierung durch die OMG (Object Management Group)
  - Hervorgegangen aus den Methoden / Notationen OMT (Rumbaugh), OOD (Booch) und OOSE (Jacobson)
  - Vereinheitlicht und standardisiert als UML seit 1996
  - Konzeptioneller Neuentwurf als Version 2 – verabschiedet Ende 2004 (752 Seiten Spezifikation der Modellierungssprache; ohne OCL, XMI, MOF)
  - Aktuelle Version 2.5.1 (seit Dez 2017, https://www.omg.org/spec/UML/2.5.1/)

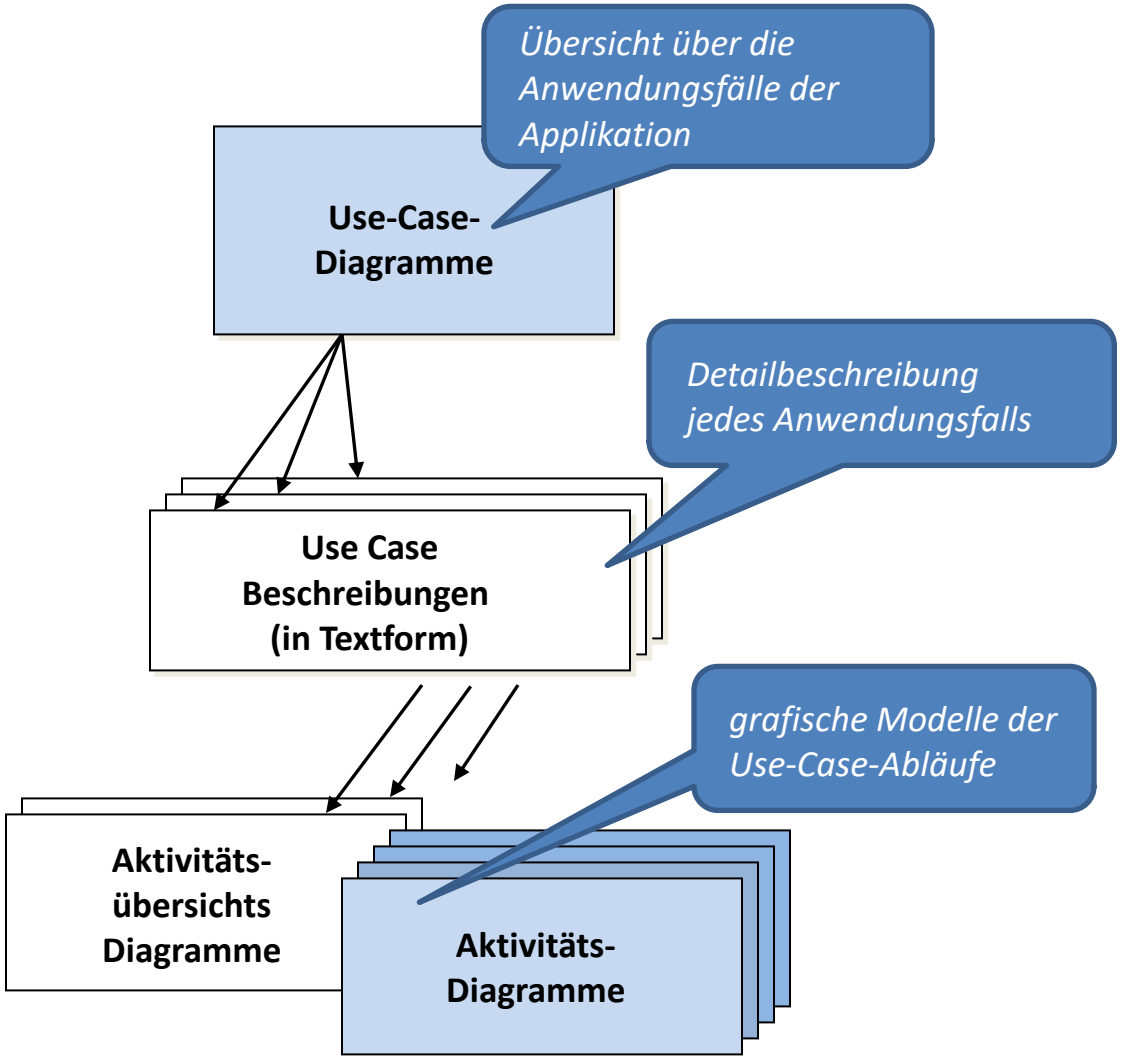
- Beispiele für freie Tools zur Modell-Erstellung
  - Visual Paradigm Online (Express-Version, https://online.visual-paradigm.com/)
  - ArgoUML (http://argouml.tigris.org/)
  - modelio (https://www.modelio.org/)
  - Violet UML Editor (http://alexdp.free.fr/violetumleditor/page.php)
  - UMLet (http://www.umlet.com/)

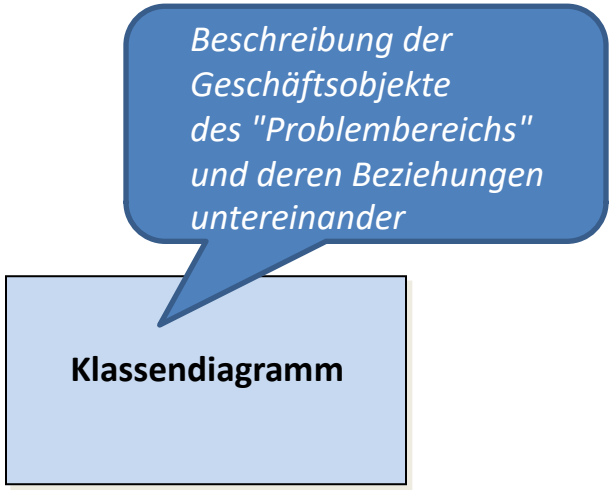
Einteilung der Diagrammformen der UML

- Die UML-Spezifikation umfasst insgesamt 13 unterschiedliche Diagrammformen mit den jeweiligen dafür vorgesehenen Modellelementen. Sie werden unterteilt in:
  - Strukturdiagramme (zur Beschreibung des statischen Modells)
    - Klassendiagramm
    - Komponentendiagramm
    - Verteilungsdiagramm
    - Kompositionsdiagramm
    - Paketdiagramm
  - Verhaltensdiagramme (zur Beschreibung des dynamischen Modells)
    - Use Case Diagramm
    - Aktivitätsdiagramm
    - Objektdiagramm
    - Zustandsautomat
    - Interaktionsdiagramme
      - Sequenzdiagramm
      - Kommunikationsdiagramm
      - Interaktionsübersichtsdiagramm
      - Timing-Diagramm

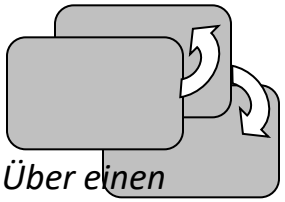
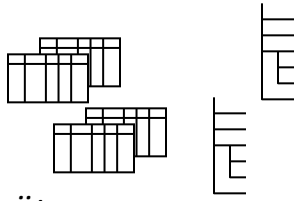
- In der objektorientierten Systemanalyse werden primär drei Diagrammformen verwendet:
  - Klassendiagramme
  - Use Case Diagramme (oder Anwendungsfall-Diagramme)
  - Aktivitätsdiagramme

UML in der Anforderungsanalyse





Zusätzlich möglich:

Beschreibung von Schnittstellen	
UI (User Interface)	System-Schnittstellen
 *Über einen prototypischen Entwurf*	 *Über Klassendiagramme*

Aufgaben

- Einführendes Kapitel 1 (Seiten 1-14) im Buch "Objektorientierte Systemanalyse" lesen sowie die Aufgaben 1 und 2 in Kapitel 1.6 bearbeiten

- Darüber hinausgehende Recherche:
  - Welche objektorientierten Programmiersprachen gibt es noch?
  - Untersuchen Sie, wie neben Python in zwei weiteren objektorientierten Sprachen die Sprachkonzepte für die Definition von Klassen (mit Attributen und Methoden), für die Vererbung und das Erzeugen von Objekten (Instanziierung) aussehen.
  - Werfen Sie einen Blick auf die Webseite der Object Management Group (www.omg.org)
    - Ein Blick in die Spezifikation der UML schadet nicht ... Nur mal Hineinstöbern ...
    - Welche weiteren Spezifikationen werden durch die OMG festgelegt?

- Praktische Übungen:
  - Programmieren Sie in Python die Klasse Girokonto analog zum Beispiel auf Seite 8. Erzeugen Sie ein Objekt der Klasse Konto und verbuchen Sie mehrere Ein- und Auszahlungen.
  - Programmieren Sie in Python die Vererbungshierarchie der Gefäße von Seite 13.
    - Fügen Sie weitere Objekte (der Klassen Glas, Fass oder Giesskanne) zur Liste hinzu.
    - Erweitern Sie die Vererbungshierarchie um eine weitere Klasse Schnapsglas. Diese soll nur die Methode leeren() mit der Ausgabe „... Prost!" umfassen. Die Methode fuellen() wird hier nicht definiert, sondern geerbt.

Vielen Dank für Ihre Aufmerksamkeit. Haben Sie noch Fragen?

IT works.

Kontakt

Dr. Nikolaus Mairon

klaus@mairon-online.de
Tel. +49 (0) 160 96678776
Skype-ID: klaus@mairon-online.de

msg nexinsure ag
Geschäftsstelle St. Georgen
Leopoldstr. 1
78112 St. Georgen

nikolaus.mairon@msg.group
Tel. +49 (0) 89 96101-3004
Mobil +49 (0) 171 9716462