

Proinformatik-Vorlesung

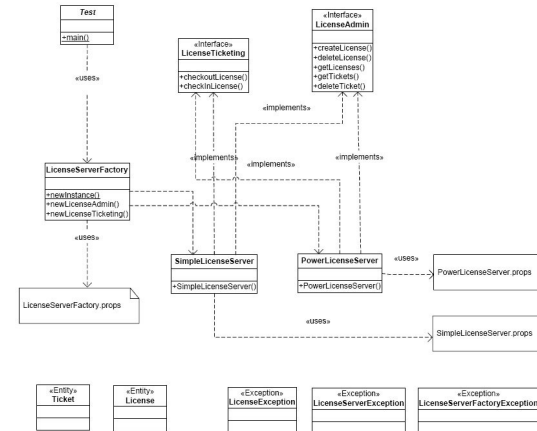
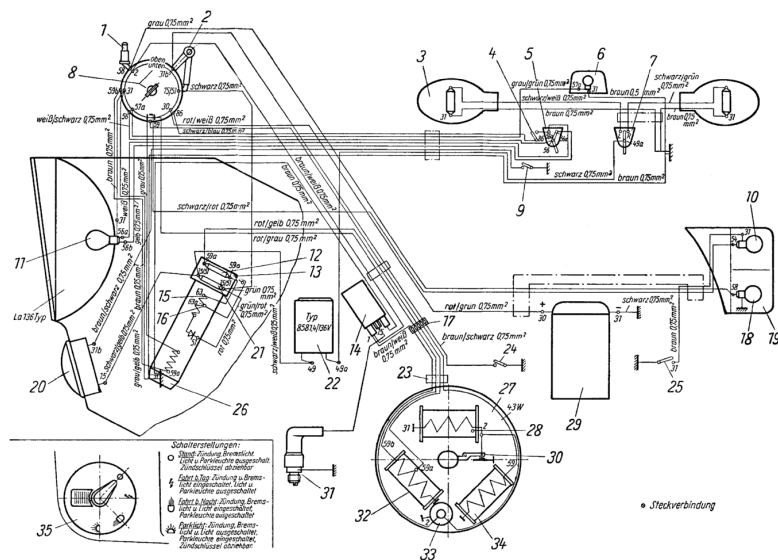
Objektorientierte Programmierung

Freie Universität Berlin



Dr. Marco Block-Berlitz
Sommersemester 2012

Aspekte der Softwaretechnik



Übersicht zum Vorlesungsinhalt

zeitliche Abfolge und Inhalte können variieren

Aspekte der Softwaretechnik

- Reifegrad der Softwaretechnik
- Analysephase
- Anforderungen, Machbarkeit, Dokumentation
- Entwurfsmuster OOA
- Entwurfsphase
- Architekturen
- Entwurfsmuster OOD/OOE
- Implementierungsphase
- Framework versus Bibliotheken
- Integrations- und Testphase
- Hoare-Kalkül, TDD
- Inbetriebnahme, Rollout und Wartung
- Projektbeispiele und Live-Projekt Maschinelles Erkennen



[9] Greching T., Bernhart M., Breiteneder R., Kappel K.: Softwaretechnik – Mit Fallbeispielen aus realen Entwicklungsprojekten, Pearson-Studium, 2010



[10] Balzert H.: UML 2 kompakt, 2. Auflage, Spektrum Verlag 2005



[11] Freeman E., Freeman E., Sierra K., Bates B.: Entwurfsmuster von Kopf bis Fuß, O'Reilly Verlag, 2005



Wie reif ist die Softwaretechnik (nach [9])?

Softwaretechnik ist im Vergleich zu etablierten, industriellen Ingenieurdisziplinen eine relativ **junge Disziplin** (ca. 50 Jahre alt). Das Bauwesen ist beispielsweise so alt wie die Menschheit selbst. Den Buchdruck gibt es seit ca. 500 Jahren.

Disziplinen zeichnen sich unter anderem dadurch aus, dass

- ihre Standardfälle wohl definiert sind
- gut genormt
- Planung kosten- und zeitmäßig verlässlich



Wie reif ist die Softwaretechnik?

Industrialisierung bedeutet, dass eine Disziplin die Transformation von genialem Kunstwerk zum Kunsthandwerk und schließlich zum industrialisierten Produkt über einen ganzen Wirtschaftsraum und Markt hinweg so weit in mehreren Wellen hinter sich gelassen hat, dass die Disziplin genau weiß, was

- ein "Kunst- und Spezialfall" für außergewöhnliche Ausnahmekönner ist
- eine sehr komplexe Aufgabe ist, mit einschätzbaren und überschaubaren Risiken
- ein Standardfall ist, dessen Zeit- und Kostenaufwände gut prognostizierbar und dessen Risiken gering sind

Wie reif ist die Softwaretechnik?

Für die Softwaretechnik lassen sich zusammenfassen:

- Pioniere, Gründer und Erfinder gab es in den 60er und 70er Jahren
- in den 80er Jahren begann die relevante Rationalisierung und Verbreitung durch den PC in der Wissenschaft, die vorher vornehmlich Algorithmen, Datenbanken, Compiler und Programmiersprachen und jetzt zunehmend Methoden im Fokus hatte
- 80er und 90er Jahre zeugen vom Willen zur Industrialisierung mit punktuellen Erfolgen, keinesfalls aber mit einer nachweislichen Marktdurchdringung über alle Systemklassen und Projektkulturen

Industriell lässt sich festhalten:

- Der Markt ist in keiner Weise durch klare **Produktklassen** segmentiert und verdichtet
- Die Disziplin hat noch **wenig Gedächtnis** und kann Erfolg von Misserfolg nur situativ und nach längerer Rückschau identifizieren
- Die **wissenschaftliche Prüfung und Bewertung** von Innovationen funktioniert (noch) nicht.

Wie reif ist die Softwaretechnik?

Unprofessionell ist es, Fehler und Scheitern mit der Jugendlichkeit der Disziplin Softwaretechnik zu begründen:

"Die wahre Substanz von Software ist unergründlich." (Mystifizierung)

"Nur Ausnahmeköpfe können gute Software schreiben." (Glorifizierung)



Folgende überprüfbare Thesen gelten dabei:

- ordentliche Softwareprojekte sind heute absolut **mach- und kontrollierbar**
- die Substanz und Komplexität eines Softwaresystems ist fast immer **methodisch beherrschbar**
- es ist eine **Disziplin**, in der die handelnden Personen über Erfolg oder Misserfolg entscheiden
- es bedarf keiner Genies – nötig sind Erfahrung, Angemessenheit und Demut; insbesondere die **Demut der Verantwortlichen** (Management und Technik) zu erkennen, wo die eigenen Grenzen liegen und Unterstützung erforderlich ist

Wie reif ist die Softwaretechnik?

Softwaretechnik ist eine junge Disziplin und das bedeutet, dass wir einen **höheren Freiheitsgrad** für das Handeln und eine **größere Verantwortung** für den einzelnen Softwaretechniker haben.

Durchschnittliches Projektpersonal kann das Projektrisiko gegenüber gutem drastisch erhöhen. Gute Technologien und Verfahren stellen zwar notwendige, aber keine hinreichenden Bedingungen für den Projekterfolg dar.

Wie reif ist die Softwaretechnik?

Die Reifegrade der Teilthemen sollten beachtet werden und dafür die entsprechende Klassifizierung vorzunehmen:

- Welche Teile meines Projektes sind objektiv **Kunst** oder **Kunsthandwerk**?
- Welche Teile befinden sich im Bereich der guten und verlässlichen **Rationalisierung**?
- Wo liefert mir die Industrialisierung der Softwaretechnik **verlässliche, preiswerte, gesicherte Lösungen** und wo ist das Versprechen des Marktes dazu nur ein Versprechen, das nicht eingelöst werden kann?
- Wie segmentiere und projiziere ich mein Projekt angemessen auf diese **Reifegrade**?
- Für welche meiner Projektmitarbeiter gilt welche **Erfahrungsstufe** in welchem Technologiebereich?
- Bin ich mir, sind sich meine Fachexperten, dieser **Mechanismen bewusst** und gehen ihre Entscheidungs- und Planungswege damit ruhig und sicher um?

Wie reif ist die Softwaretechnik?

Wichtige Punkte für angehende Softwaretechniker:

- Hören Sie nicht auf, zu **lesen**, zu **lernen** und **neugierig** zu sein.
- Zur guten Theorie gehören praktische Erfahrungen in relevanten und realen Projekten.
- Was in einem Projekt mit 6 Personenmonaten (PM) erfolgreich ist, kann bei 30 PM scheitern.
- Wer Projekte mit maximal 30 PM angeführt und verantwortet hat, kann als Leiter bei 300 PM zum Projektrisiko werden.



Persiflag zur Softwaretechnik



Was der Kunde
erklärte



Was der Projektleiter
verstand



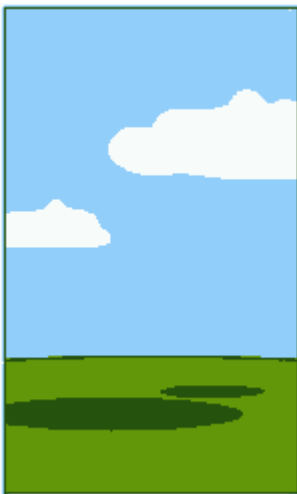
Wie es der Analytiker
entwarf



Was der Programmierer
programmierte



Was der Berater
definierte



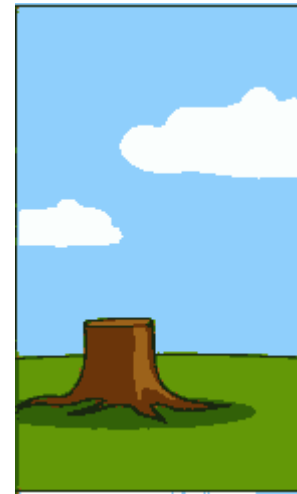
Wie das Projekt
dokumentiert wurde



Was installiert wurde



Was dem Kunden in
Rechnung gestellt
wurde



Wie es gewartet
wurde

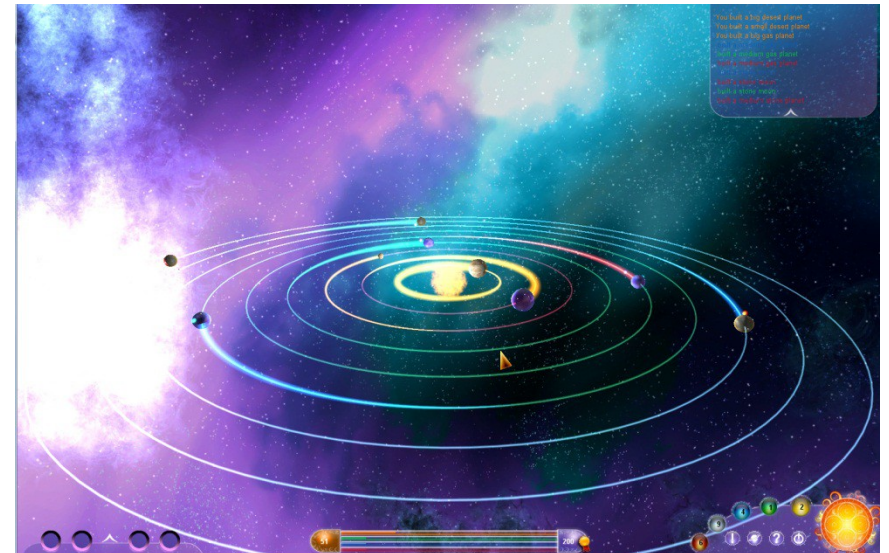


Was der Kunde
wirklich gebraucht
hätte

Phasen der Softwareentwicklung

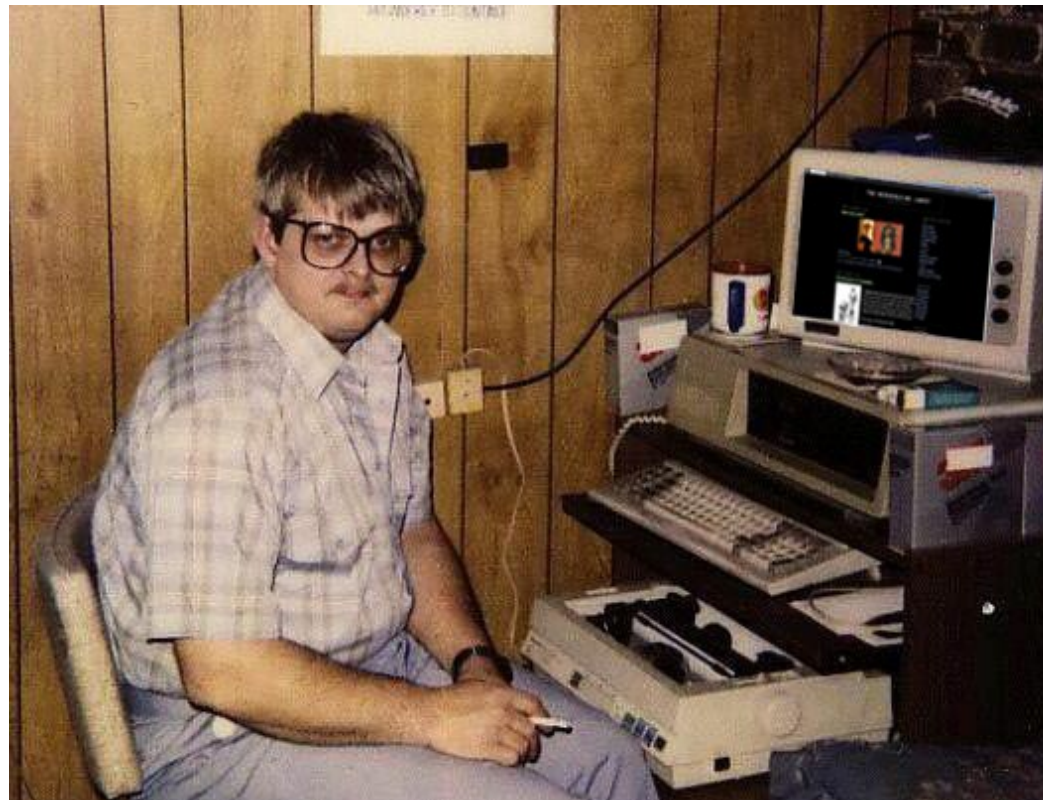
Es gibt typischerweise fünf Phasen in der Softwareentwicklung:

- 1) Analysephase (Anforderungsdokumentation, Machbarkeitsstudie)
- 2) Entwurfsphase (OOD-Dokument)
- 3) Implementierungsphase (Spezifikation)
- 4) Integrations- und Testphase (Test- und Abnahmeprotokolle)
- 5) Inbetriebnahme, Rollout und Wartung (Spezifikationspflege)



Gamedesign ist nichts für Alleingänger

GAMEDESIGN IST EIN WACHSENDER MARKT DESSEN
FINANZVOLUMEN BEREITS JETZT GRÖßER IST ALS DAS DER
FILMINDUSTRIE



Wichtige Rollen im Gamedesign



Narrator



User-Interface-Designer



Tester



Komponist



Leveldesigner



3D-Artist



Programmierer



Video-Artist



Concept-Artist



Texturierer

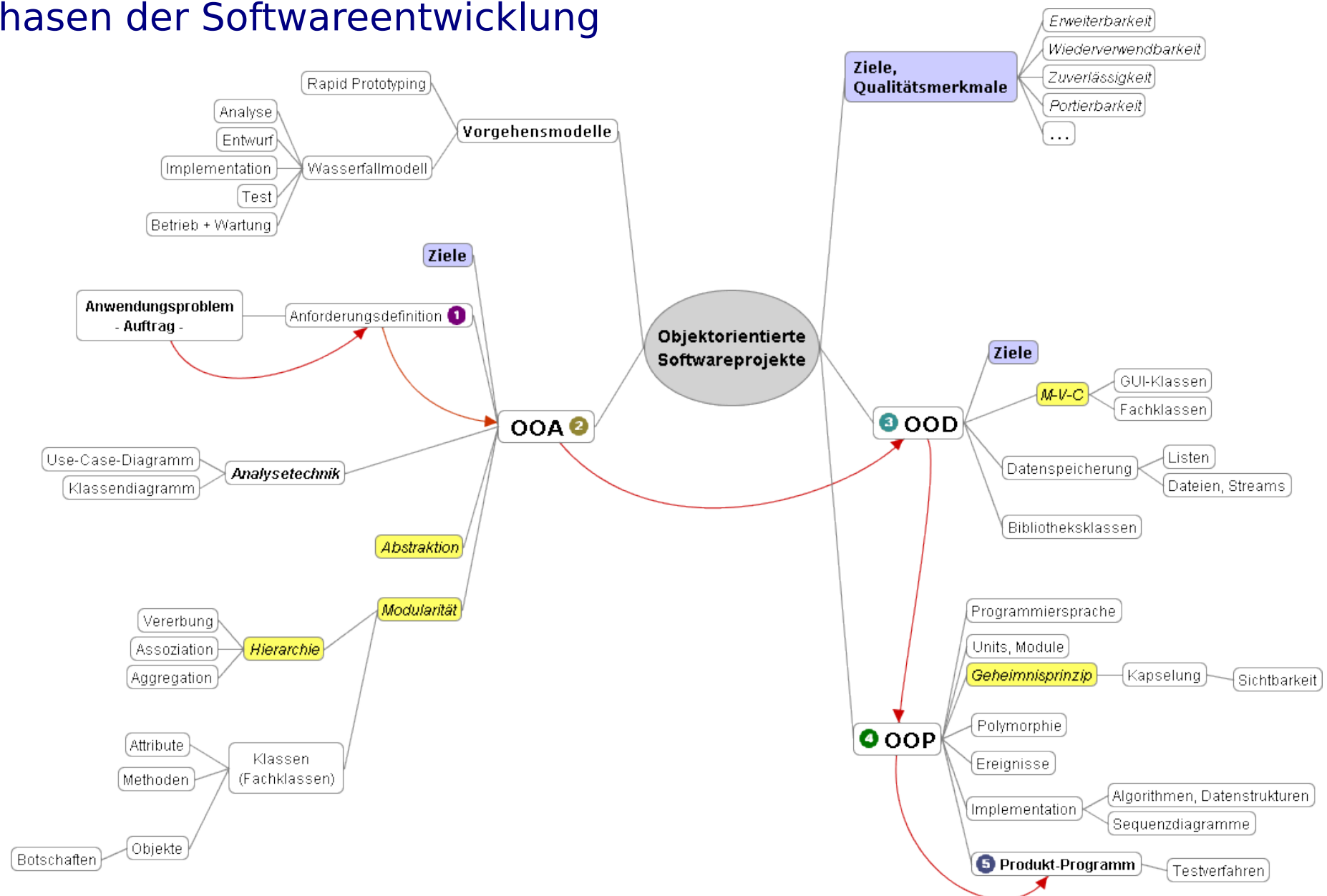


Entwickler



Producer

Phasen der Softwareentwicklung



Analysephase



Analyse



Entwurf



Implementierung



Integration
Test



Inbetriebnahme, Rollout,
Wartung

Phase 1: Analyse

Die Analyse hat in erster Linie das Ziel, die **Vision** und den **Scope** eines Softwareprojekts zu definieren. Hier werden die wichtigsten Entscheidungen getroffen, wobei Fehlentscheidungen zu sehr teuren Nachbesserungen führen können.

Anschließend werden alle technischen Fragestellungen evaluiert, alle **Stakeholder** identifiziert und deren Anforderungen erhoben und dokumentiert. Soziale Faktoren können dabei die Analyse erschweren (z.B. Glaubensgrundsätze, versteckte Regeln, ...).

- Vision:** Zweck und Nutzen, die das System erfüllen soll
- Scope:** Umfang des Systems (auch Nicht-Ziele definieren)
- Stakeholder:** Alle am Projekt beteiligten Personen (Kunde, Nutzer, Entwickler, ...)

Anforderungen

Für alle Systemanforderungen muss in der Analysephase die **Volatilität** bestimmt werden. Stabile Anforderungen stellen die Basis für die Grobarchitektur dar.

Anforderungen sollten mit **Attributen** versehen und ständig aktualisiert werden:

- Vollständig

- Eindeutig definiert

- Verständlich beschrieben

- Atomar

- Einheitlich dokumentiert

- Notwendig

- Nachprüfbar

- Rück- und vorwärtsverfolgbar

- Priorisiert

Machbarkeitsstudie

Bei jedem Projekt muss gründlich analysiert werden, ob es überhaupt machbar ist. Dazu wird eine **Machbarkeitsstudie** durchgeführt. Diese kann komplett analytisch oder auch durch prototypische Umsetzungen der kritischen Systemteile erfolgen.

Zu den Inhalten gehören dabei auch:

- Kostenanalyse

- Konkurrenzsituation

- Rechtliche Prüfung

- Akzeptanzanalyse

- Make-or-buy

- Grobarchitektur

Anforderungsmodellierung und -dokumentation

Die aktuell relevanteste, visuelle Modellierungssprache, die aus verschiedenen Diagrammtypen besteht, ist **UML** (Unified Modeling Language).

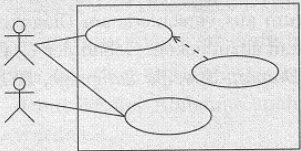
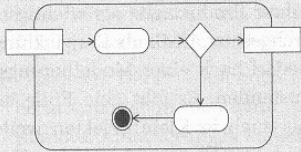
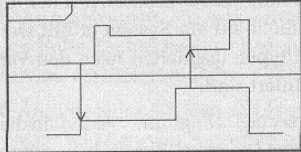
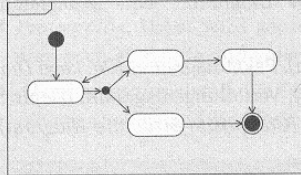
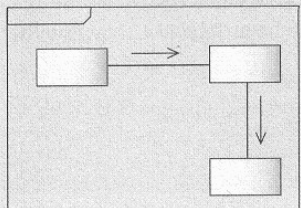
Es gibt Verhaltensmodelle (z.B. Usecase-Diagramm) und Strukturmodelle (z.B. Klassen-Diagramm).

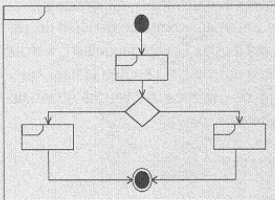
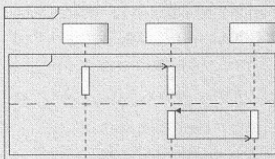
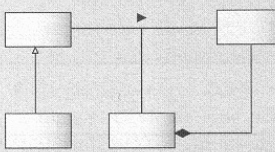
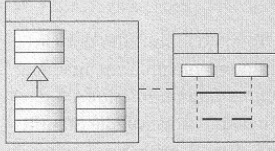
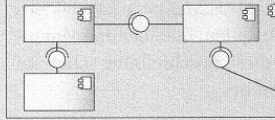
Die Anforderungen werden in der **Anforderungsdokumentation** (auch Spezifikation, Software-Requirements-Specification kurz SRS) zusammengetragen und sind Teil der Vertragsgrundlage für ein Softwareprojekt. Dabei werden alle Anforderungen eindeutig bezeichnet, z.B. mit Nummern versehen. Im deutschsprachigen Raum gibt es dafür ein Lasten- und ein Pflichtenheft.

Das **Lastenheft** ist im Besitz des Auftraggebers und definiert die Anforderungswünsche. Das **Pflichtenheft** wird vom Auftragnehmer geführt und definiert die Anforderungsbasis für das Entwicklungsvorhaben.

Eine stabile Zwischenversion der Anforderungsanalyse wird als **Baseline** bezeichnet.

Unified Modeling Language – kurz gefasst I (aus [9])

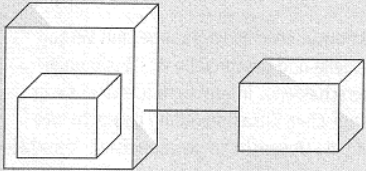
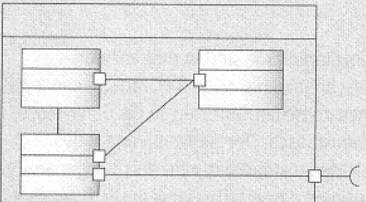
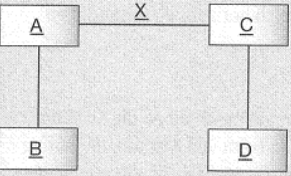
UML-Diagramm	Merkmale / Beschreibung
Anwendungsfalldiagramm (Usecase Diagram) 	Ein Anwendungsfalldiagramm stellt die Beziehungen zwischen Akteuren und Anwendungsfällen aus Sicht der Akteure/Stakeholder dar. Es ist ein Verhaltensmodell und modelliert somit keine Ablaufbeschreibung. Anwendungsfalldiagramme werden zur Modellierung der spezifizierten Anforderungen an ein System eingesetzt.
Aktivitätsdiagramm (Activity Diagram) 	Das Aktivitätsdiagramm gehört zu den Verhaltensmodellen. Es stellt Aktivitäten und deren Verbindungen grafisch dar. Eine Aktivität ist ein elementarer Schritt innerhalb eines Programmablaufes. Das Aktivitätsdiagramm wird verwendet, um den Ablauf von Anwendungsfällen zu beschreiben.
Zeitverlaufdiagramm (Timing Diagram) 	Zeitverlaufdiagramme erlauben eine zeitliche Sicht auf dynamische Aspekte und Interaktionen eines Systems. Sie bilden Interaktionen und deren zeitliche Abläufe ab. Zeitverlaufdiagramme gehören zu den Verhaltensmodellen.
Zustandsdiagramm (Statechart Diagram) 	Zustandsdiagramme gehören zu den Verhaltensmodellen. Sie beschreiben mögliche Systemzustände und bilden Zustandsänderungen sowie deren auslösende Ereignisse ab. Es wird ein endlicher Zustandsautomat dargestellt, der sich zur Laufzeit in einer Menge endlicher Zustände befindet.
Kommunikationsdiagramm (Communication Diagram) 	<i>Kommunikationsdiagramme beschreiben die zwischen Objekten übertragenen Nachrichten und stellen Assoziationen zwischen Klassen dar. Die zeitliche Abfolge von Nachrichten wird mittels Sequenzausdrücken beschrieben. Sequenzausdrücke definieren, welche Nachrichten Vorgänger sind und welche parallel ausgetauscht werden. Kommunikationsdiagramme gehören ebenfalls zu den Verhaltensmodellen.</i>

UML-Diagramm	Merkmale / Beschreibung
Interaktion-Übersichtsdiagramm (Interaction Overview Diagram) 	Interaktionsübersichtsdiagramme sind Verhaltensmodelle und beschreiben das Zusammenspiel verschiedener Interaktionen, indem sie in einen logischen Zusammenhang gebracht werden, um die Übersicht zu gewährleisten. Verschiedene Interaktionsdiagramme werden mit der Notation des Aktivitätsdiagramms verbunden.
Sequenzdiagramm (Sequence Diagram) 	Sequenzdiagramme sind Verhaltensmodelle und beschreiben die zeitliche Ordnung von Interaktionen zwischen Objekten sowie deren Nachrichtenaustausch. Der zeitliche Verlauf des Nachrichtenaustausches wird durch sogenannte Lebenslinien dargestellt. Dabei verläuft die Zeitachse senkrecht von oben nach unten. Objekte werden durch senkrechte, die ausgetauschten Nachrichten durch horizontale Lebenslinien beschrieben.
Klassendiagramm (Class Diagram) 	Klassendiagramme beschreiben die Struktur eines Systems in Form von Klassen, die untereinander in Beziehung stehen. Eine Klasse ist eine Zusammenfassung gleichartiger Objekte in Bezug auf Eigenschaften und Fähigkeiten.
Paketdiagramm (Package Diagram) 	Paketdiagramme bilden eine Teilmenge von Klassendiagrammen ab. Sie gruppieren Elemente eines Systems in sogenannte Pakete, um dadurch die Anzahl der Beziehungen und Abhängigkeiten zwischen Elementen zu verringern.
Komponentendiagramm (Component Diagram) 	Komponentendiagramme beschreiben die Anordnung von Softwarekomponenten sowie ihre Abhängigkeiten und Schnittstellen. Softwarekomponenten sind wiederverwendbare Systemteile, die jeweils eine spezifizierte Aufgabe erfüllen.

VM

SM

Unified Modeling Language – kurz gefasst II (aus [9])

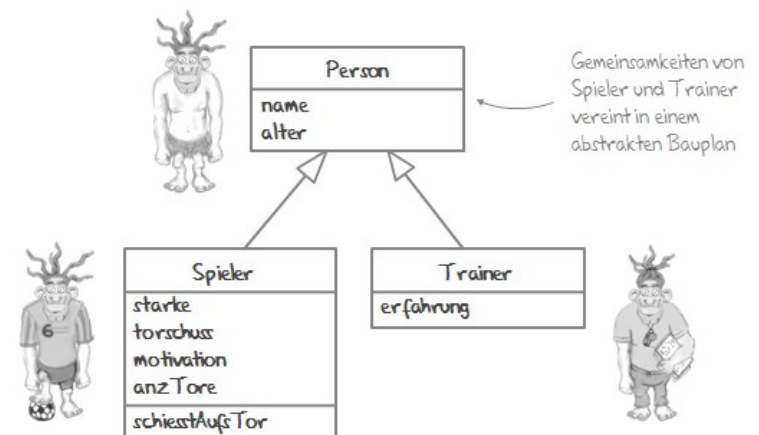
UML-Diagramm	Merkmale / Beschreibung
<p>Verteilungsdiagramm (Deployment Diagram)</p> 	<p>Verteilungsdiagramme beschreiben die Konfiguration einer Hard- und Softwareumgebung (Knoten) in Bezug auf Komponenten und deren Verteilung auf diesen. Es wird modelliert, welche Komponenten auf welchen Knoten laufen, wie diese konfiguriert werden und welche Abhängigkeiten existieren.</p>
<p>Kompositions-Struktur-Diagramm (Composite Structure Diagram)</p> 	<p>Kompositionsstrukturdiagramme beschreiben die interne Struktur eines Klassifizierers (Classifier) sowie seine Interaktionen zu anderen Systemkomponenten.</p>
<p>Objektdiagramm (Object Diagram)</p> 	<p>Objektdiagramme beschreiben die statische Struktur eines Systems zu einem bestimmten Zeitpunkt in Bezug auf die Belegung der Attribute und Ausprägung von Assoziationen.</p>

Objekte in der Analyse identifizieren

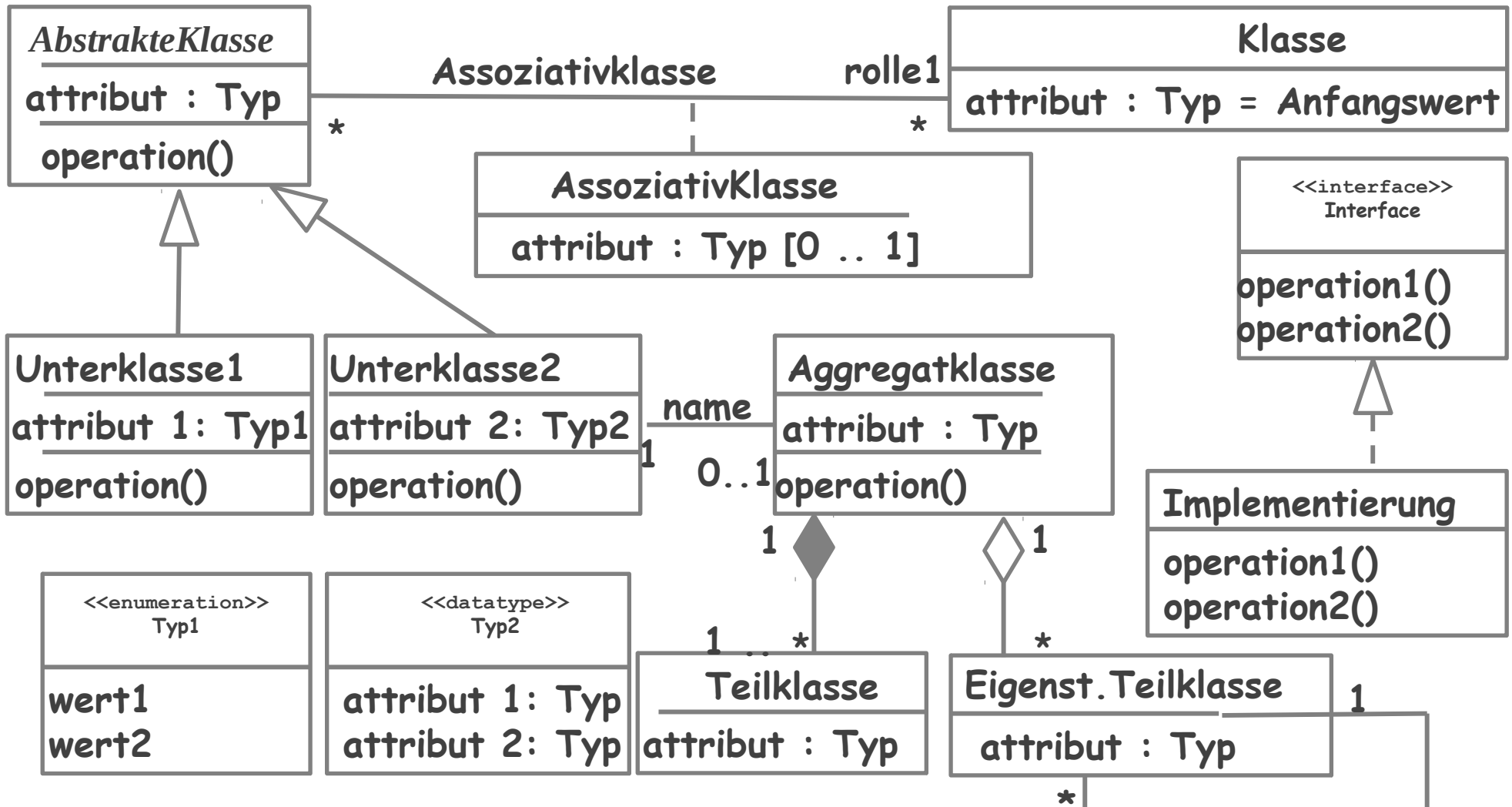
Wir versuchen alle **Objekte** und **Objektbeziehungen** der "realen Welt" zu identifizieren und stellen uns die Frage: Wie soll die Software genutzt werden?. Welche Funktionalität soll wem zur Verfügung gestellt werden? **Was** geschieht **warum** mit Objekten?

Objekte sind gedankliche oder reale Einheiten mit Zustand und Funktionalität. Es gibt **Funktionen**, die den inneren Objektzustand verändern oder Wirkungen auf andere Objekte haben. Dann bestehen sogenannte **Objektbeziehungen**.

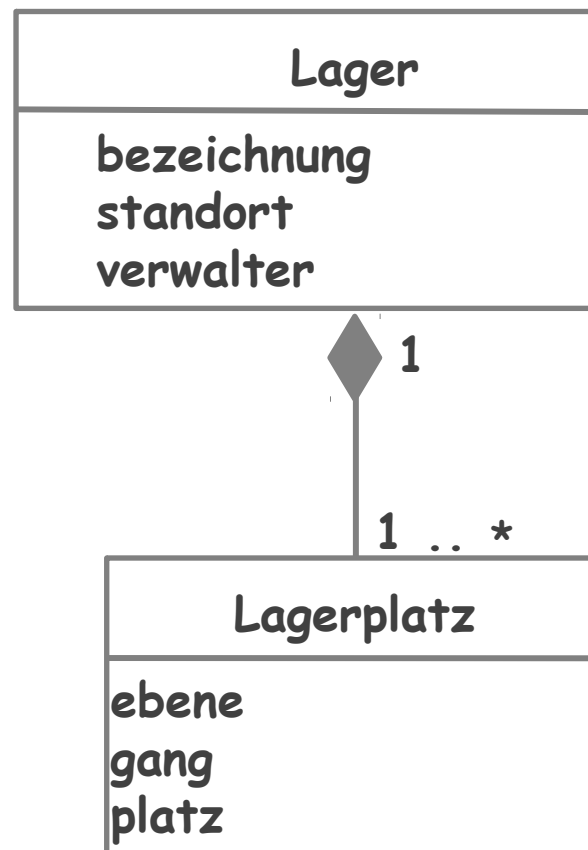
Der Zustand eines Objekts wird repräsentiert durch **Zustandsvariablen** und **Attributen**. Die Attribute sollten nach außen nicht sichtbar sein (**Geheimnisprinzip**, **Kapselung**), sondern nur über Methoden gesetzt oder gelesen werden (**Selektoren/get-set-Methoden**).



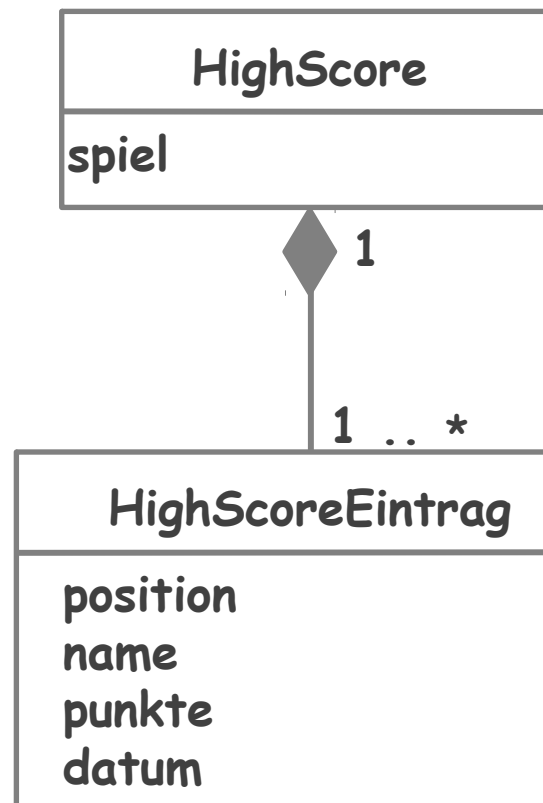
UML Klassendiagramm [10 (erweitert)]



Analysemuster Liste [10]

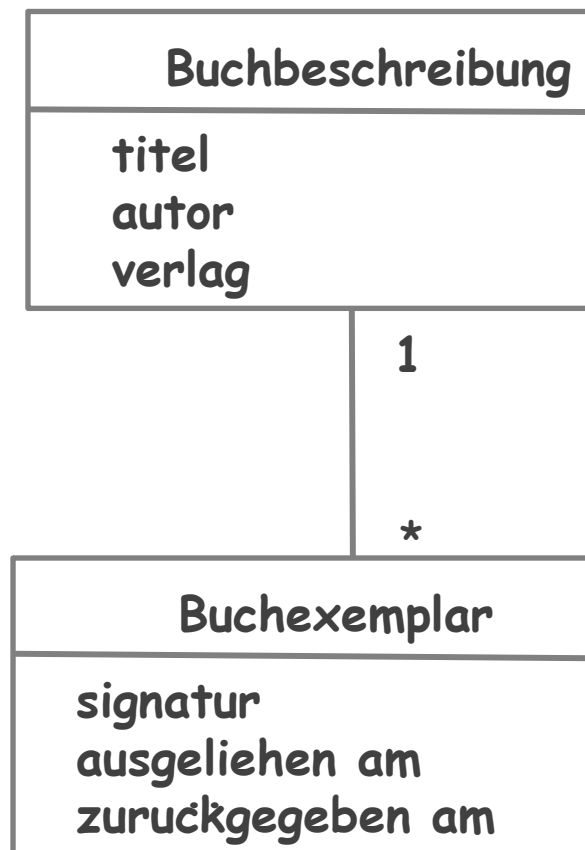


Analysemuster Liste

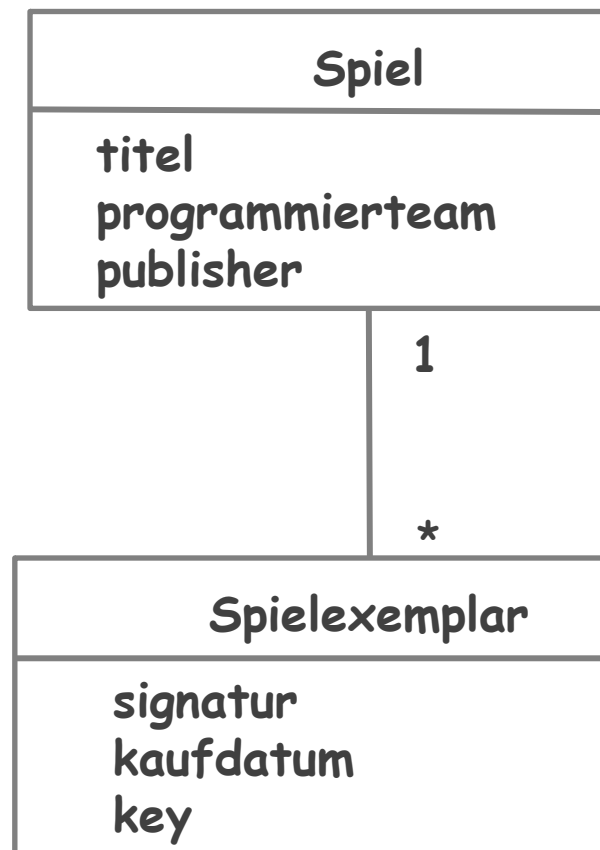


Position	Username	Punkte
1	sauch	80
2	mhertler	69
2	msteiner	69
4	Bruder Tack	65
5	jlutz	64
5	Markus	64
7	rdoster	63
8	mdaum Mueller	61
9	sstoll	60
10	Tanny	59
11	sauch2	56
11	skurfess	56
13	C00ky	54
13	smack	54
15	Michael Mutzel	50
15	Robson	50
17	amack	44
17	Drucker	44
19	jfrasch	43
20	mgrunewald	8

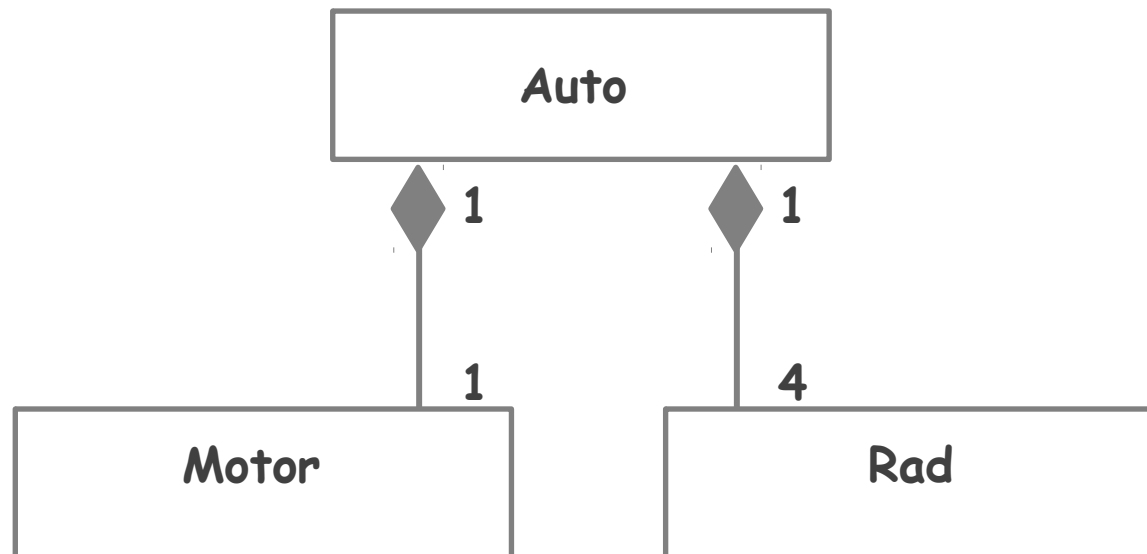
Analysemuster Exemplartyp [10]



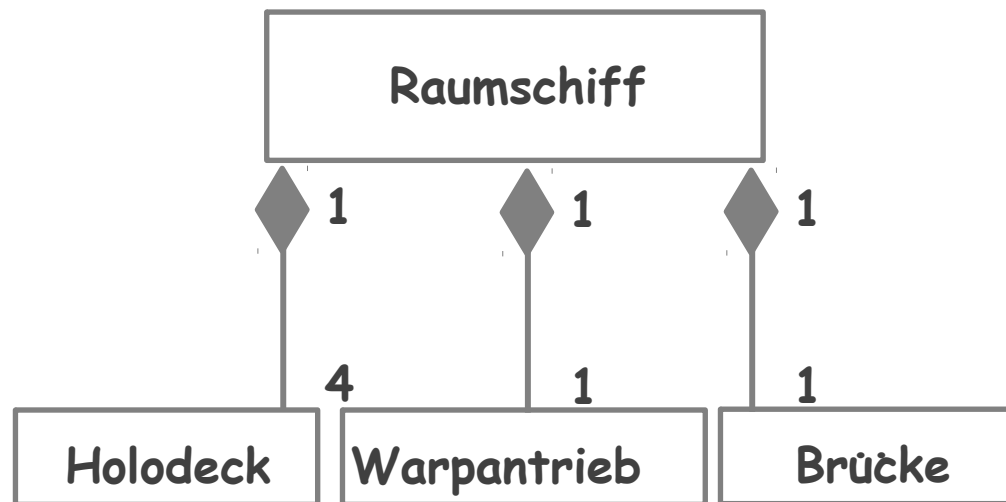
Analysemuster Exemplartyp



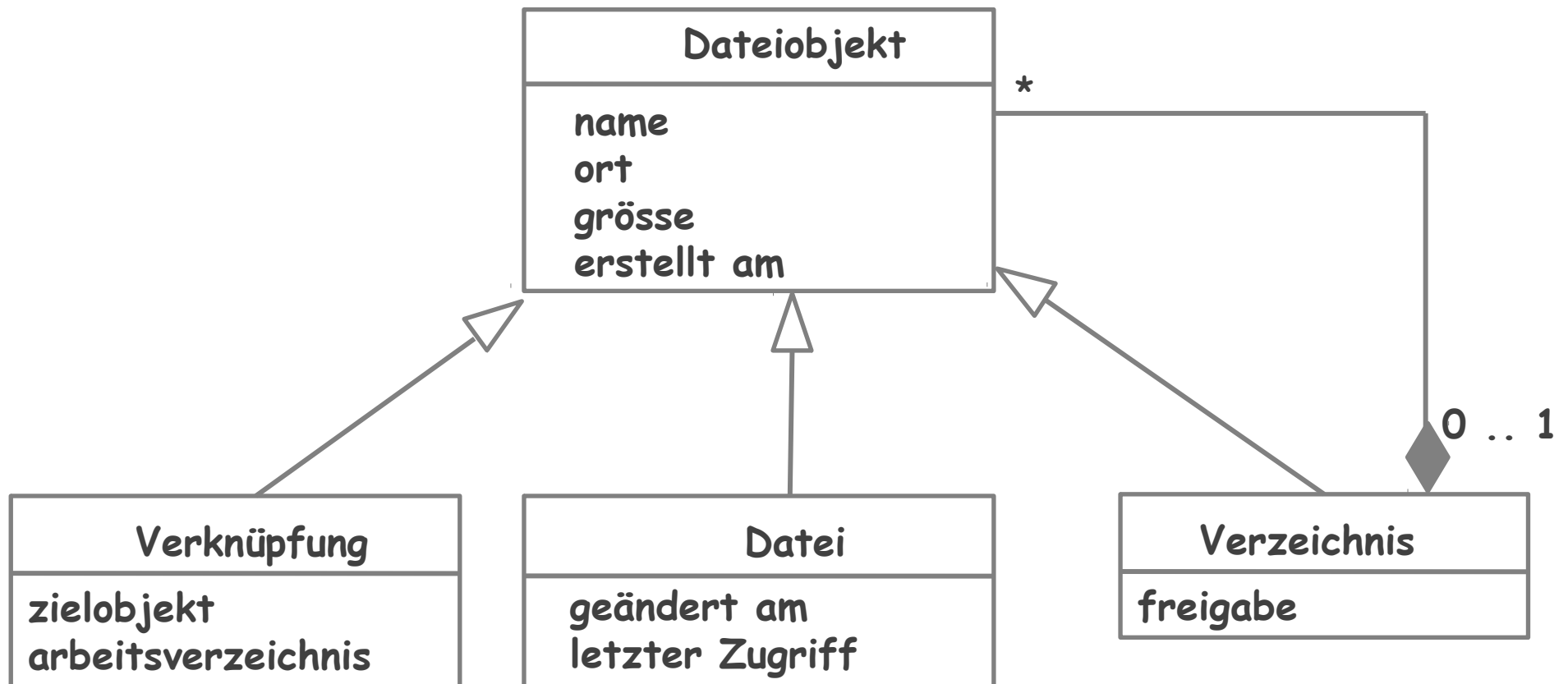
Analysemuster Baugruppe [10]



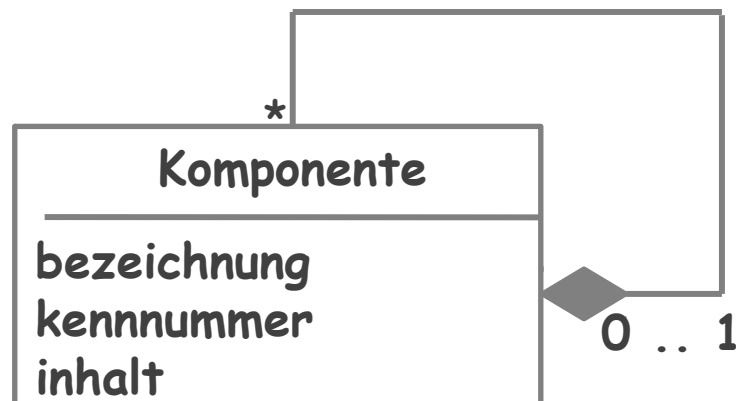
Analysemuster Baugruppe



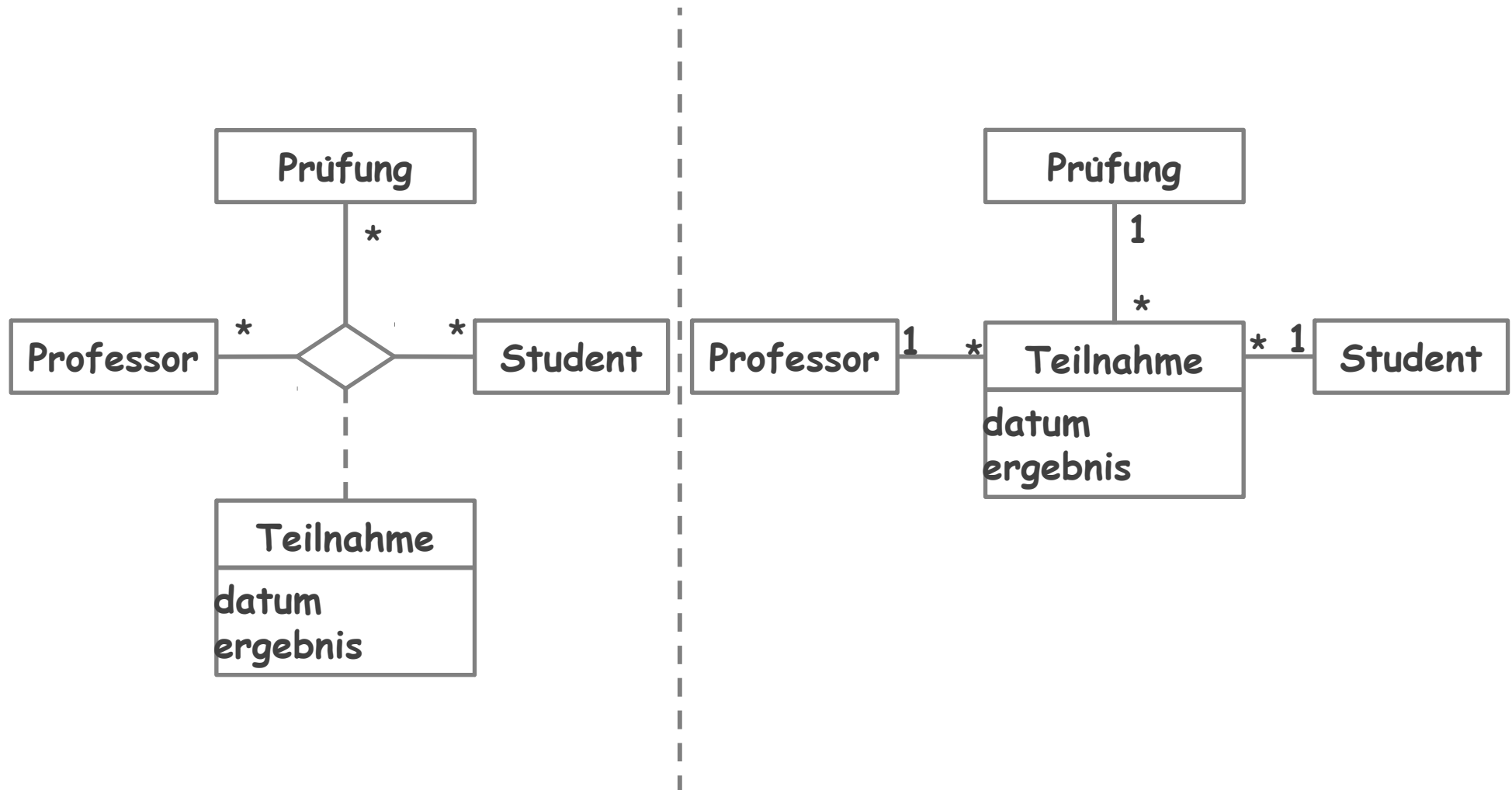
Analysemuster Stückliste [10]



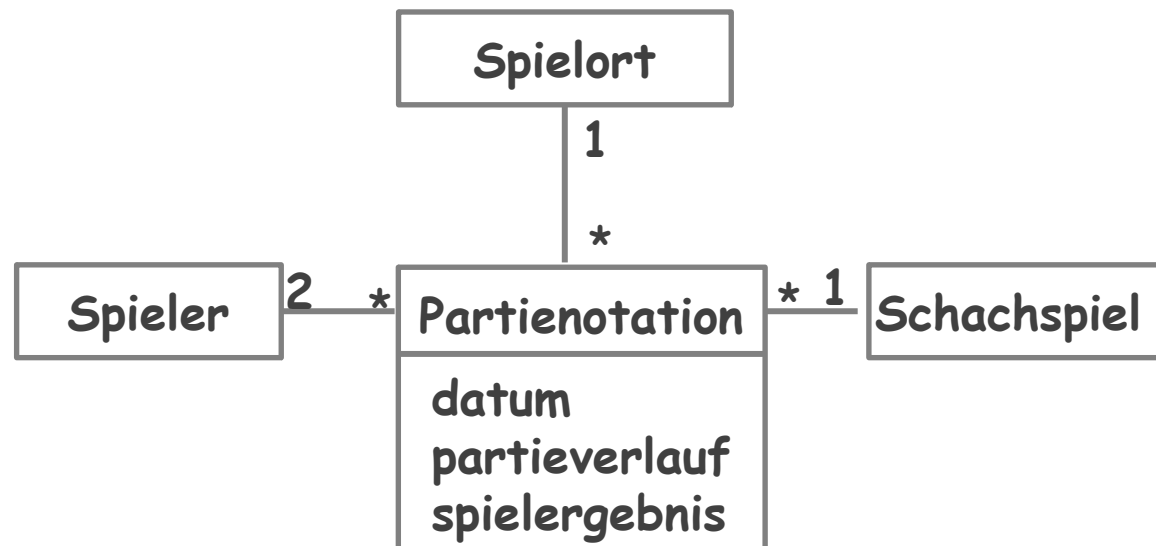
Analysemuster Stückliste (Sonderfall)



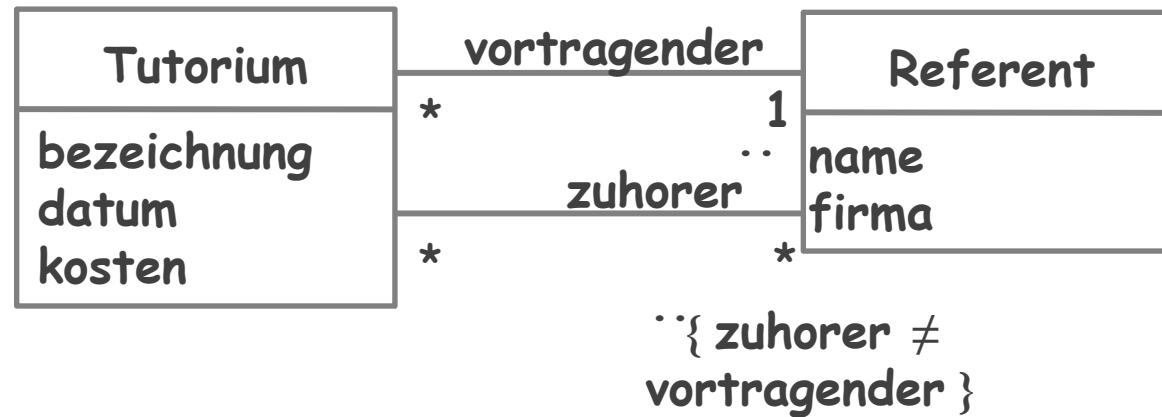
Analysemuster Koordinator [10]



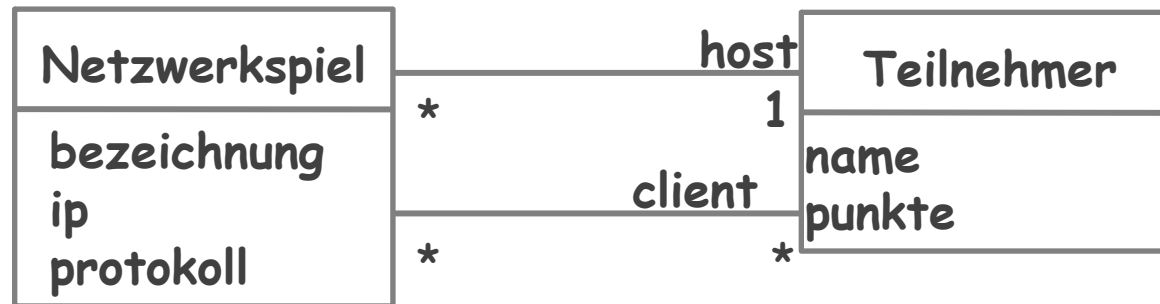
Analysemuster Koordinator



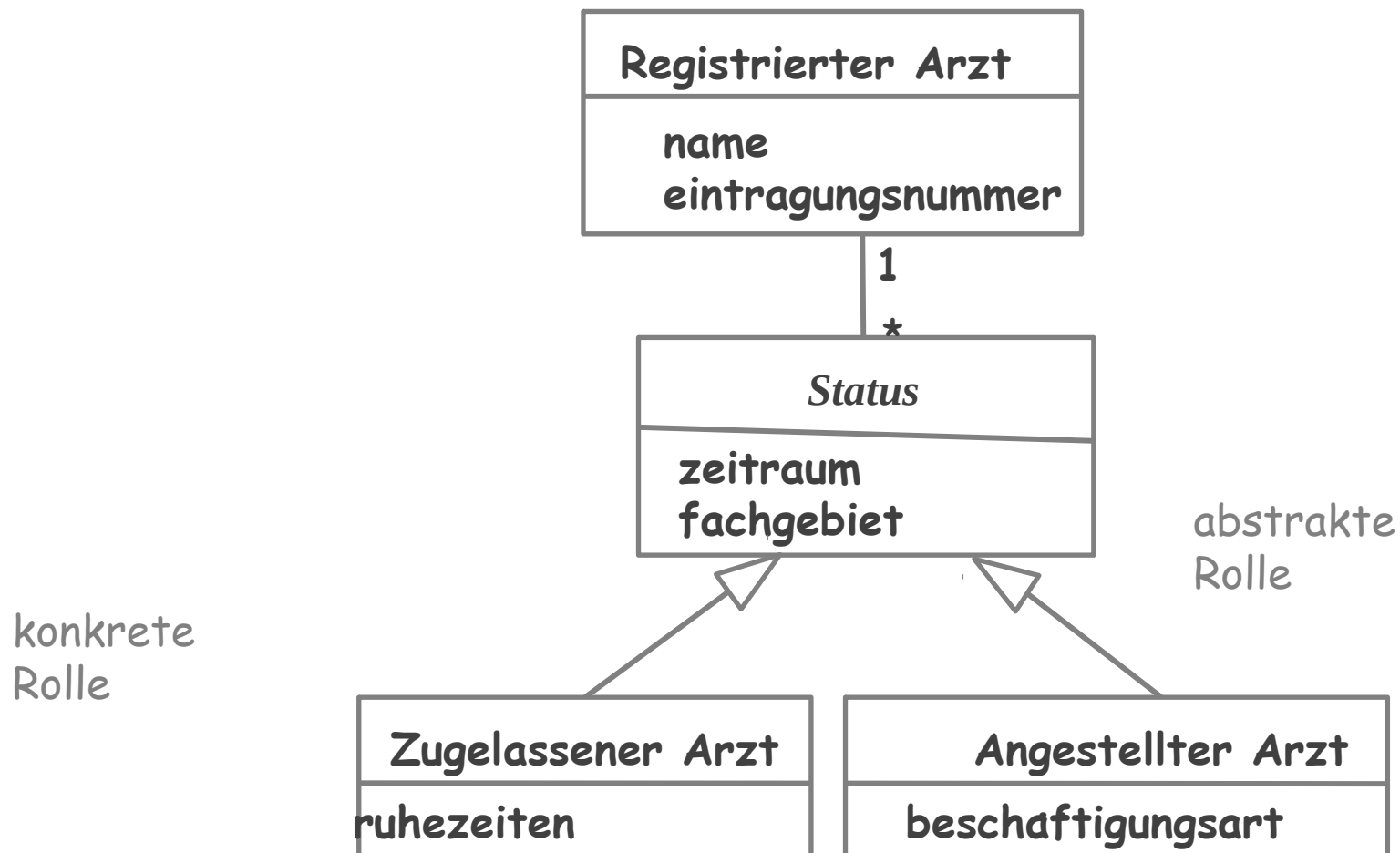
Analysemuster Rollen [10]



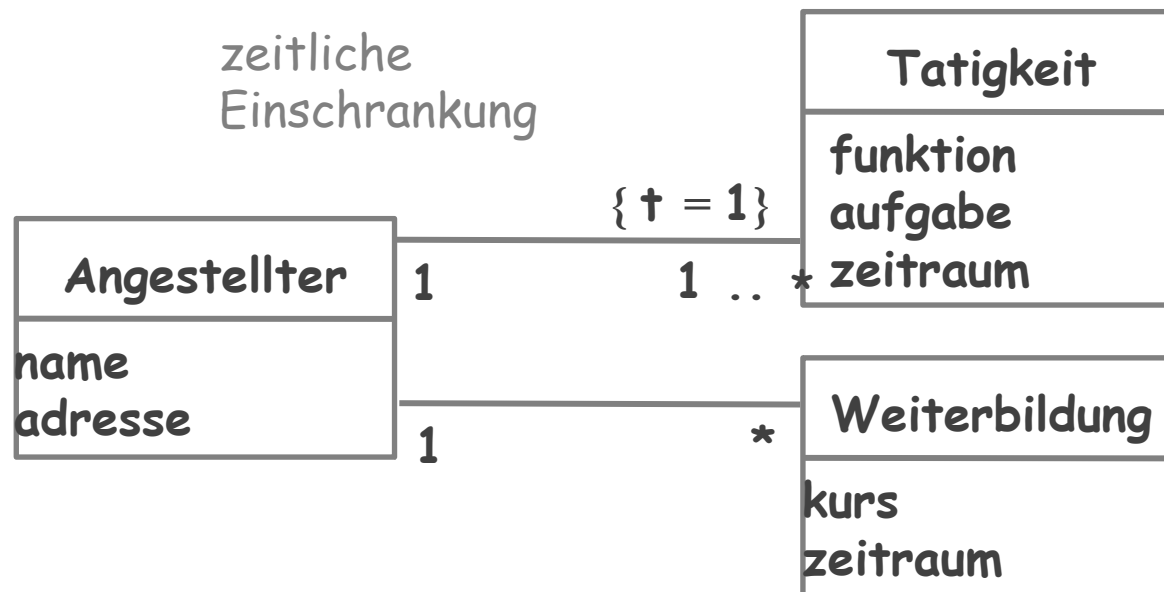
Analysemuster Rollen



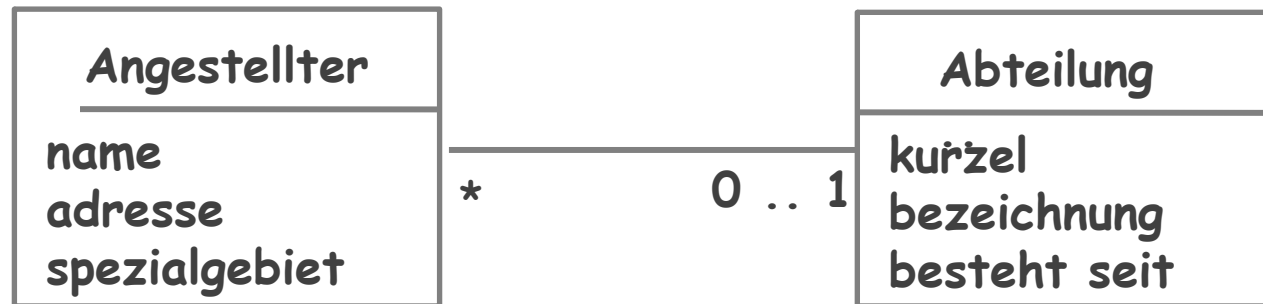
Analysemuster wechselnde Rollen [10]



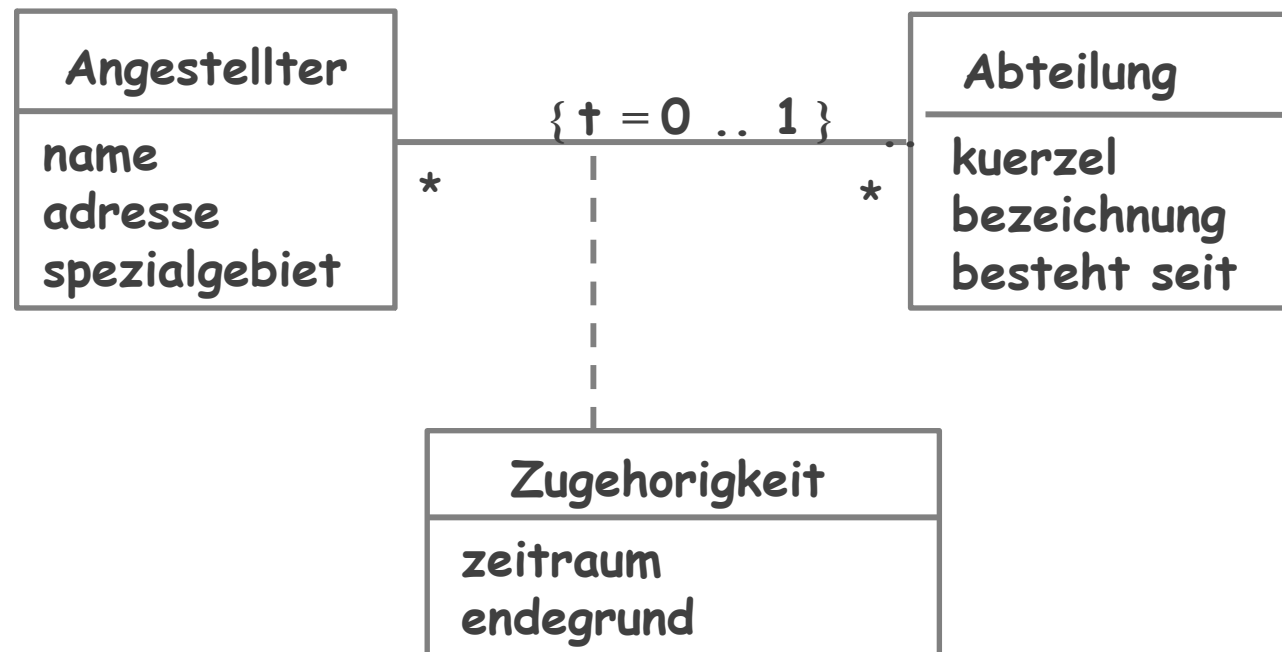
Analysemuster Historie [10]



Analysemuster Gruppe [10]



Analysemuster Gruppenhistorie [10]



Entwurfsphase



Analyse



Entwurf



Implemen-
tierung



Integration
Test



Inbetrieb-
nahme, Rollout,
Wartung

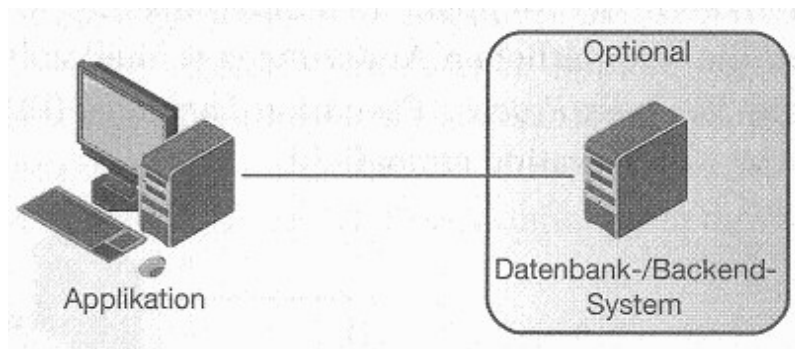
Phase 2: Entwurf

Ziele des Entwurfs sind es, das System in funktional getrennte Blöcke zu teilen (**Modularisierung**) und ein System zu entwickeln, das die Anforderungen des Kunden optimal beschreibt.

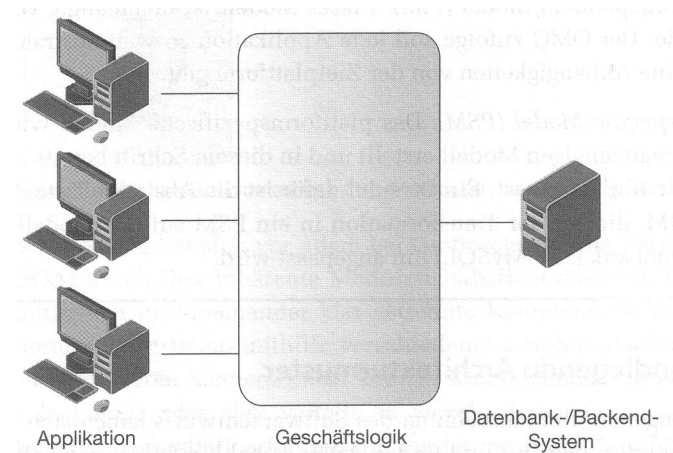
Der Entwurf ist die Architektur im Kleinen und die Softwarearchitektur die Architektur im Großen.

Grundlegende Architekturmuster

2-Schichten-Architektur



3-Schichten-Architektur



Objektorientiertes Design(OOD)

Im Objektorientierten Design/Entwurf (OOD/OOE) werden die Diagramme und Modelle der OOA übernommen und erweitert. Ziel ist es, eine möglichst genaue Abbildung des später zu implementierenden Systems abzubilden.

Die jetzt relevanten Diagramme sind Klassen-, Zustands-, Sequenz- und Komponenten-diagramme, die wieder in UML abgebildet werden.

Erstellung der Makroarchitektur:

- Klassen und Objekte werden auf jeder Abstraktionsebene identifiziert





- Die Semantiken und Beziehungen zwischen den Klassen werden festgelegt

- Schnittstellen und Implementationen der Klassen werden spezifiziert

Also Festlegung einer Architektur für die Implementierung und Policies festlegen für die Systemkomponenten. Auch die Fehlerbehandlung und die Testplanung gehören in die Entwurfsphase.

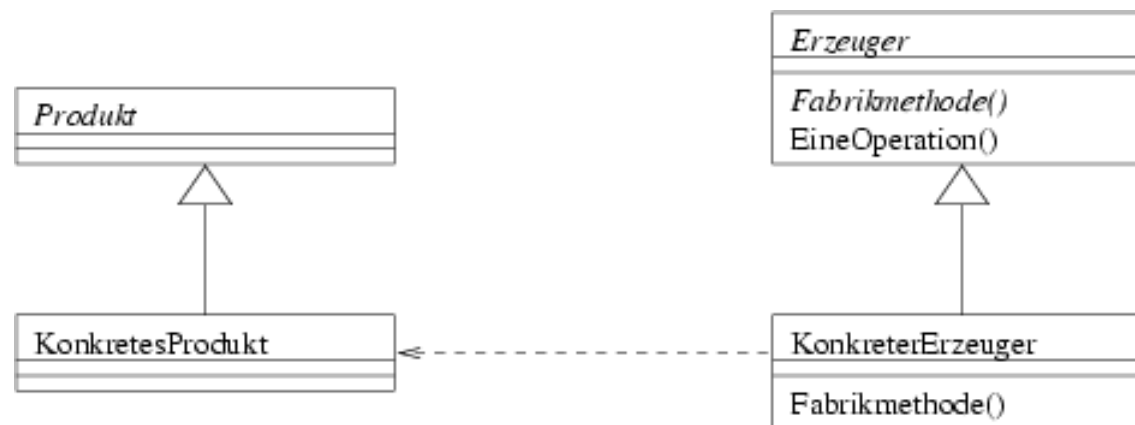
Entwurfsmuster

Für die erfolgreiche Erstellung eines strukturierten Quellcodes sind die seit langer Zeit bekannten Entwurfsmuster (Design Patterns) eine gute Grundlage:

Erzeugende Muster 	Strukturelle Muster 	Verhaltensmuster 	Weitere Muster 
Abstract Factory (Abstrakte Fabrik)	Adapter	Chain of Responsibility (Zuständigkeitskette)	Business Delegate
Builder (Erbauer)	Composite (Kompositum)	Command (Kommando)	Data Access Object
Factory Method (Fabrikmethode)	Bridge (Brücke)	Interpreter	Data Transfer Object (Datentransferobjekt)
Prototype (Prototyp)	Decorator (Dekorierer)	Iterator	Dependency Injection
Singleton (Einzelstück)	Facade (Fassade)	Mediator (Vermittler)	Inversion of Control
	Flyweight (Fliegengewicht)	Memento	Model View Controller
	Proxy (Stellvertreter)	Null Object (Nullobjekt)	Model View Presenter
		Observer (Beobachter)	Plugin
		State (Zustand)	Fluent Interface
		Strategy (Strategie)	
		Template Method (Schablonenmethode)	
		Visitor (Besucher)	

Factory-Pattern

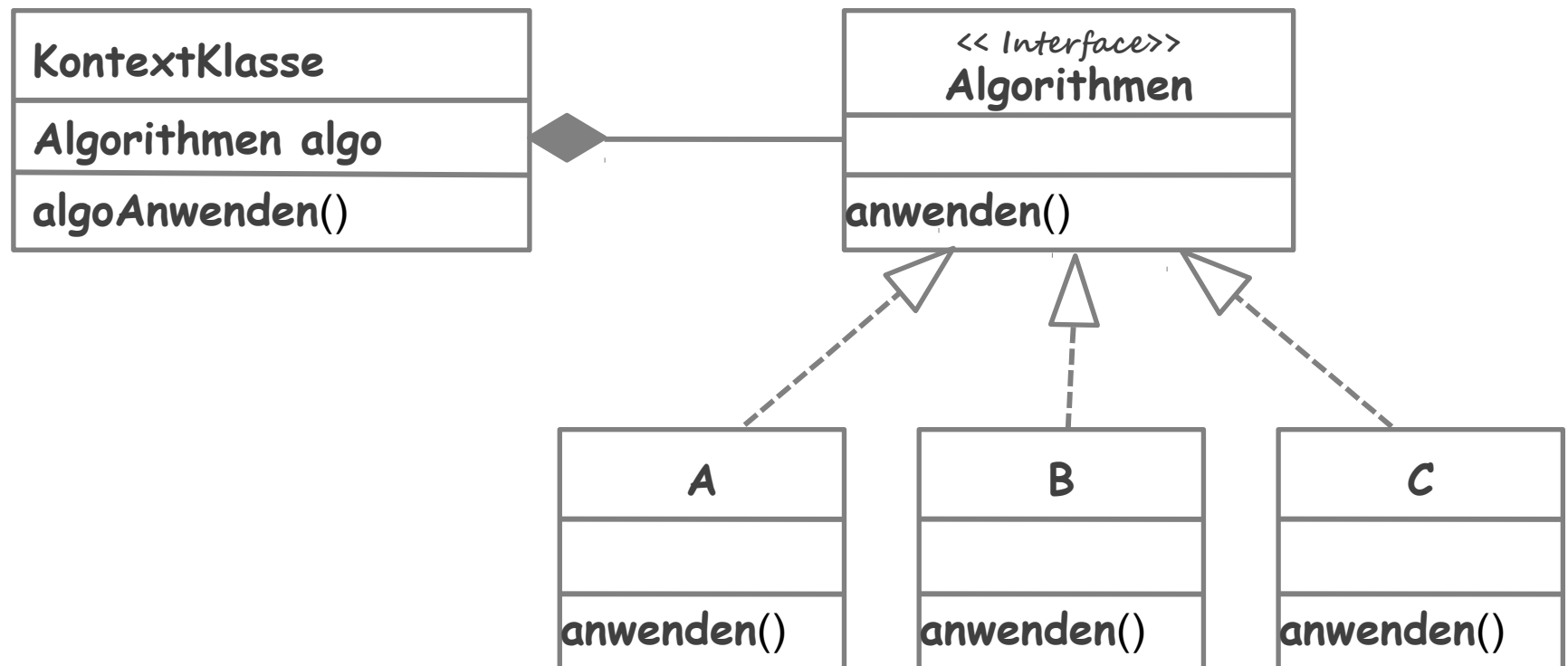
Das Muster beschreibt, wie ein Objekt durch Aufruf einer Methode anstatt durch direkten Aufruf eines Konstruktors erzeugt wird.



Fabrikmethoden entkoppeln ihre Aufrufer von Implementierungen konkreter Produktklassen. Das ist insbesondere wertvoll, wenn Frameworks sich während der Lebenszeit einer Applikation weiterentwickeln - so können zu einem späteren Zeitpunkt Instanzen anderer Klassen erzeugt werden, ohne dass sich die Applikation ändern muss.

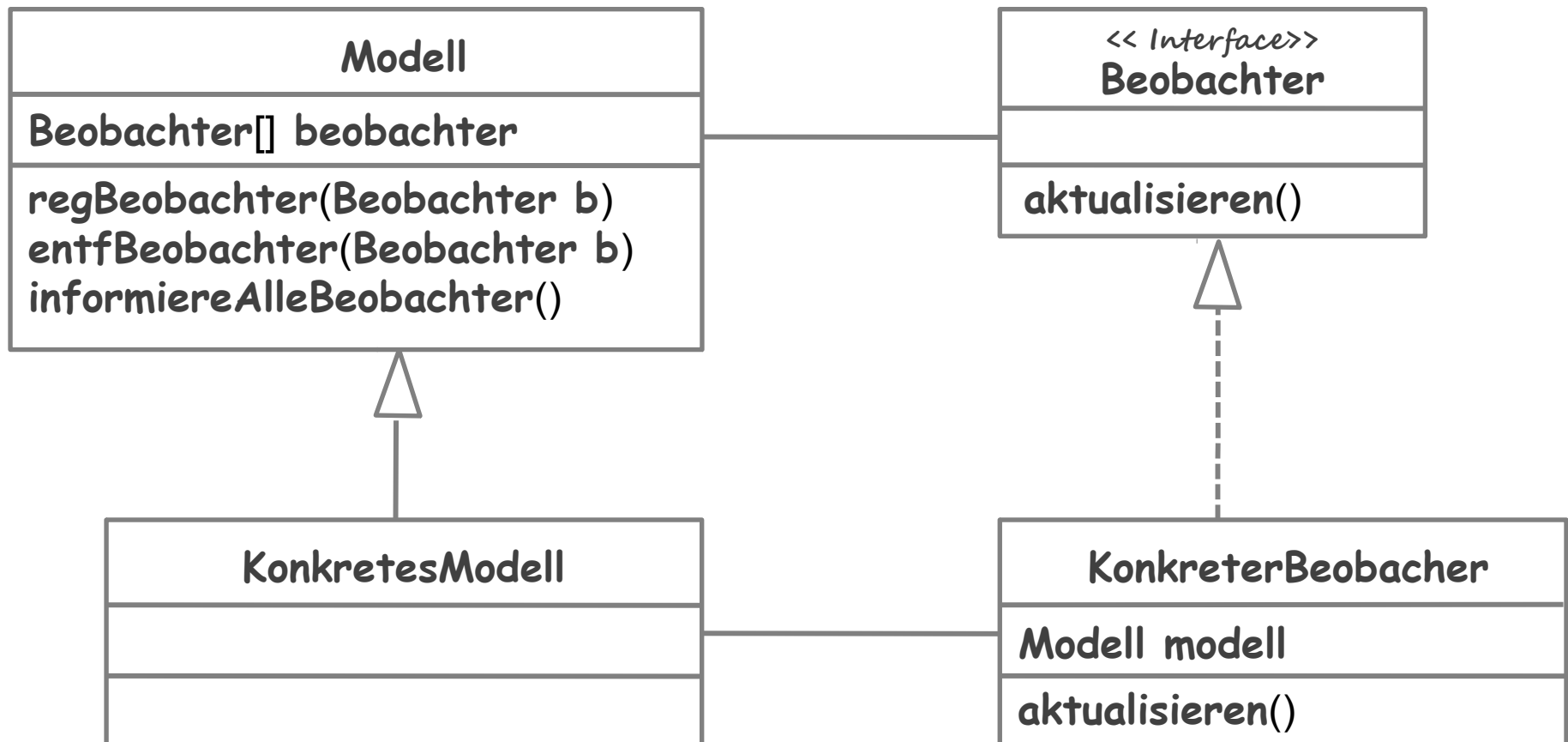
Strategy-Pattern

Die „Hat-ein“- kann der „Ist-ein“-Beziehung oft überlegen sein. Ziehen Sie Komposition deshalb der Vererbung vor...



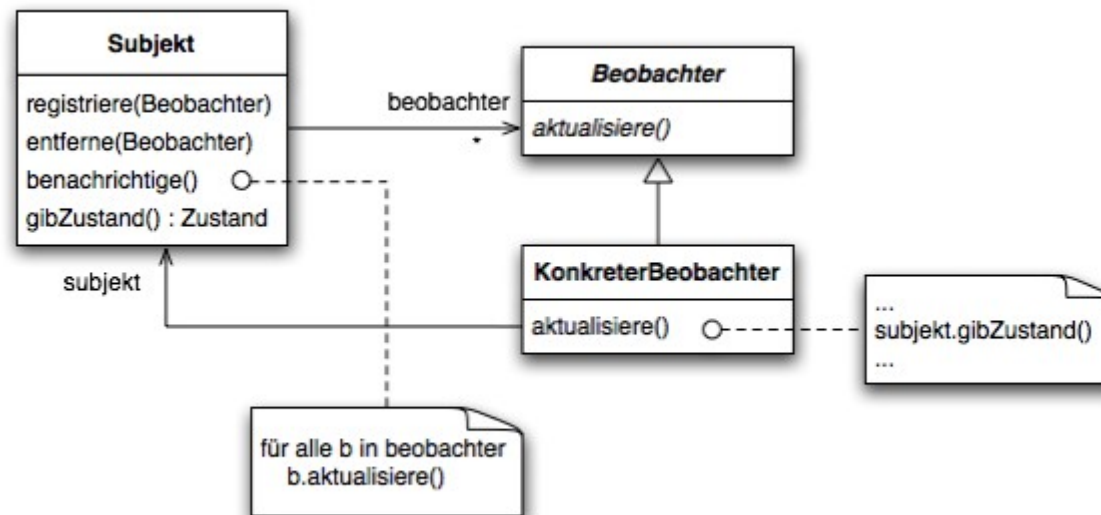
Das Strategy-Pattern definiert eine Familie von Algorithmen, kapselt sie einzeln und macht sie austauschbar.

Observer-Pattern



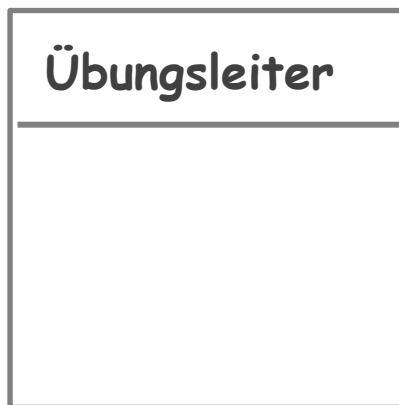
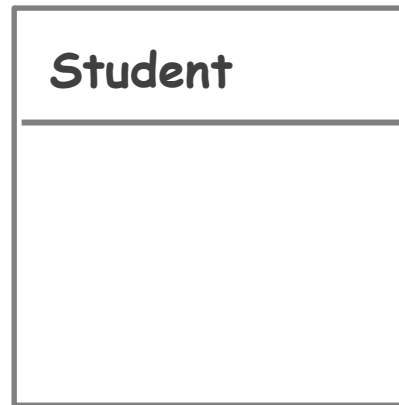
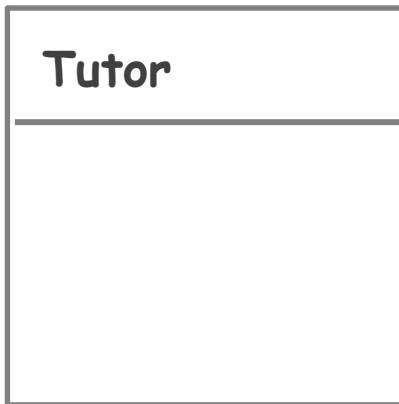
Observer

Es dient zur Weitergabe von Änderungen an einem Objekt an von diesem Objekt abhängige Strukturen.



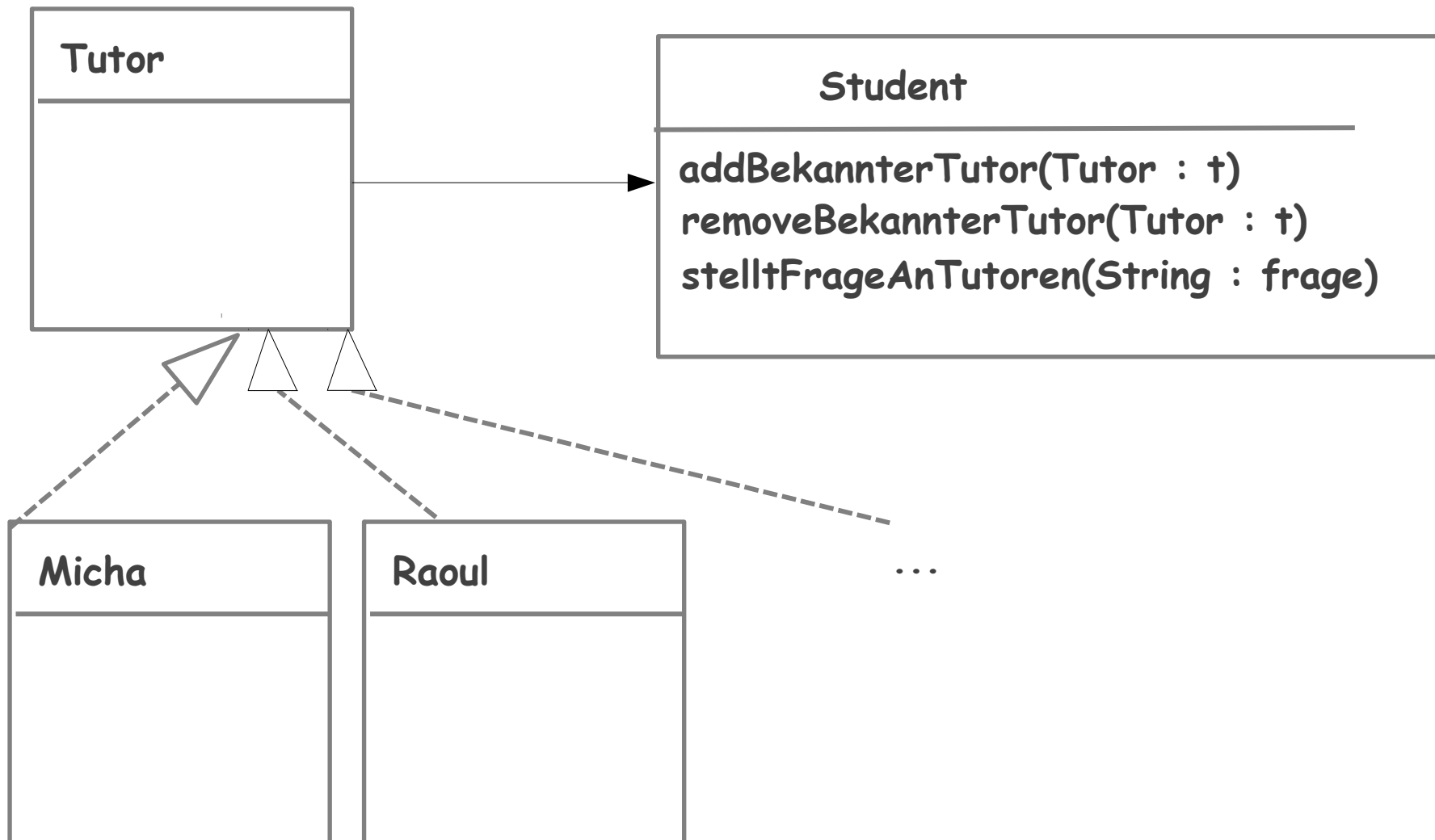
Das beobachtete Objekt braucht keine Kenntnis über die Struktur seiner Observer zu besitzen, sondern kennt diese nur über die Observer-Schnittstelle. Ein abhängiges Objekt erhält die Änderungen automatisch. Änderungen am Objekt führen bei großer Observeranzahl zu hohen Änderungskosten.

Proinfbeispiel - Modell

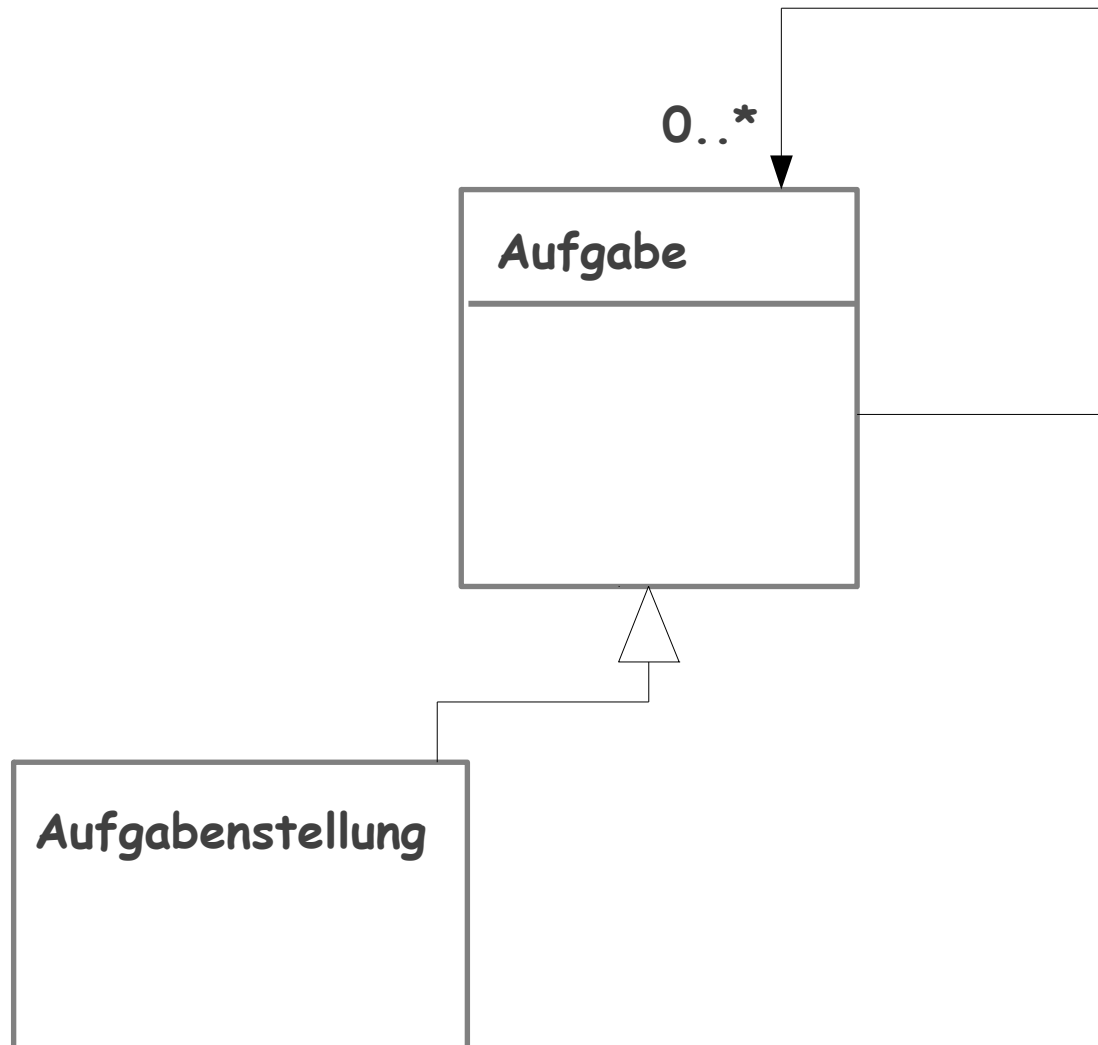


Proinfbeispiel – Modell - Tutor

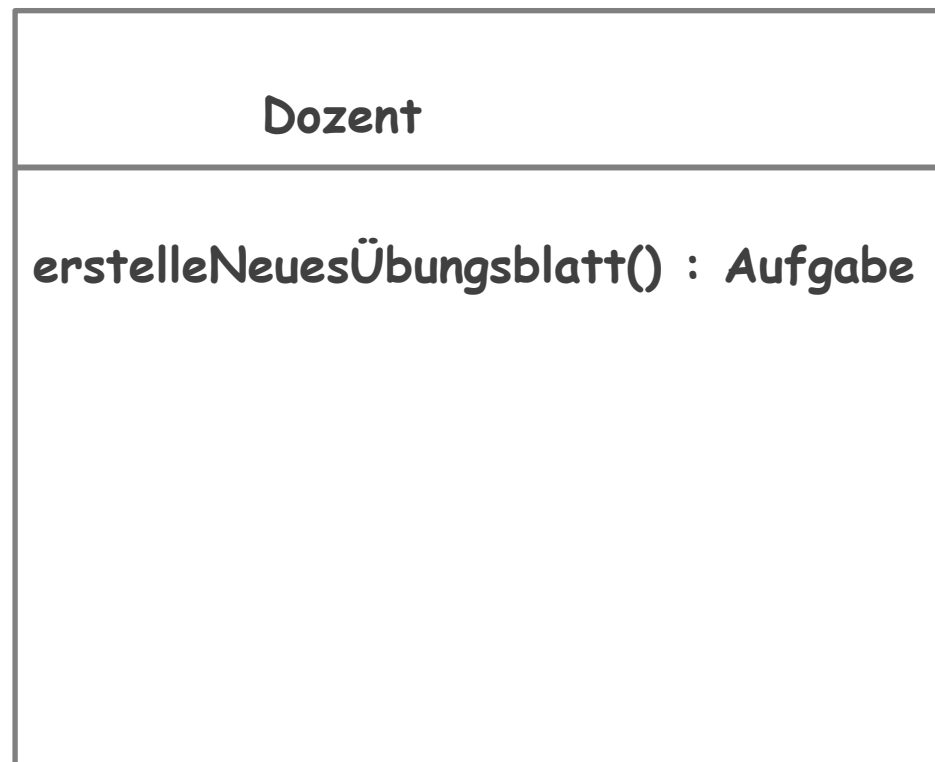
Was ist das für ein Muster?



Was ist das für ein Muster?



Was ist das für ein Muster?



Was ist das für ein Muster?

Übungsleiter

```
+ static getÜbungsleiter() : Übungsleiter  
- static übungsleiter : Übungsleiter  
- Übungsleiter();
```