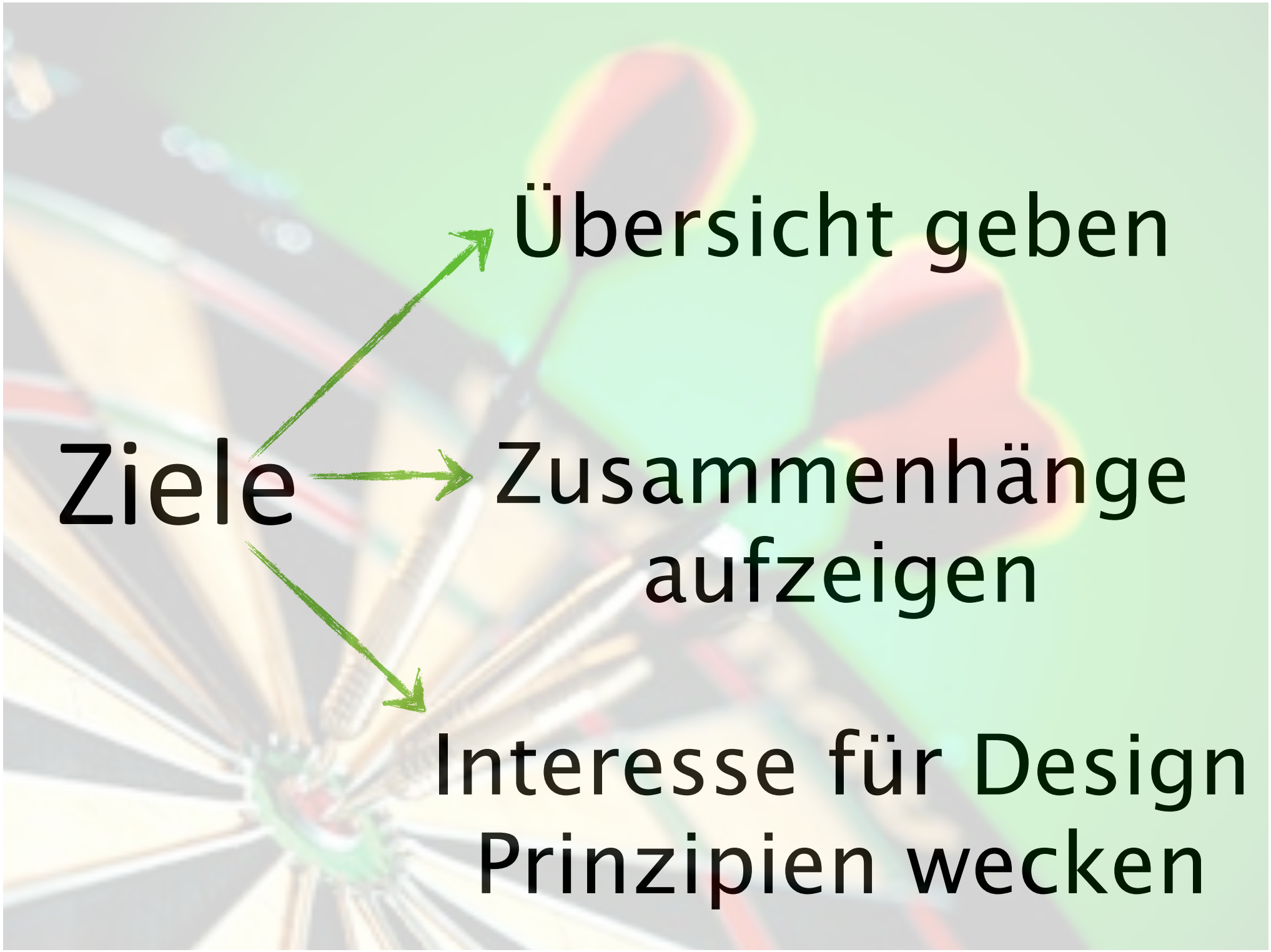
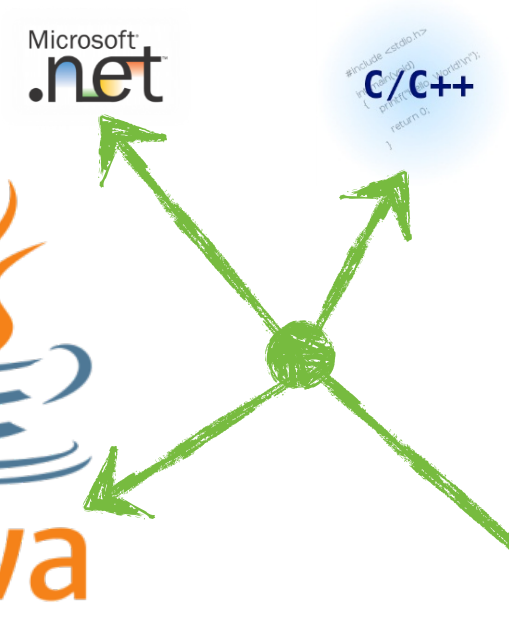
Die Macht, die uns umgibt

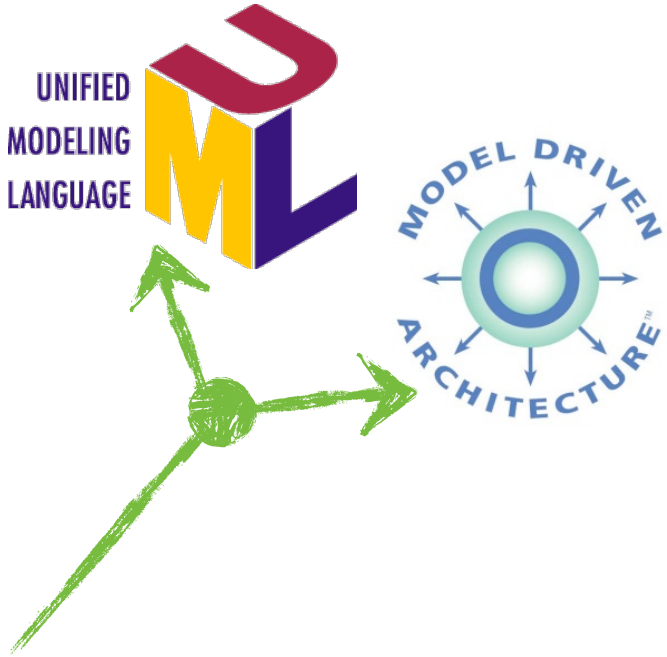
Design Prinzipien

Schneller und besser Software entwickeln

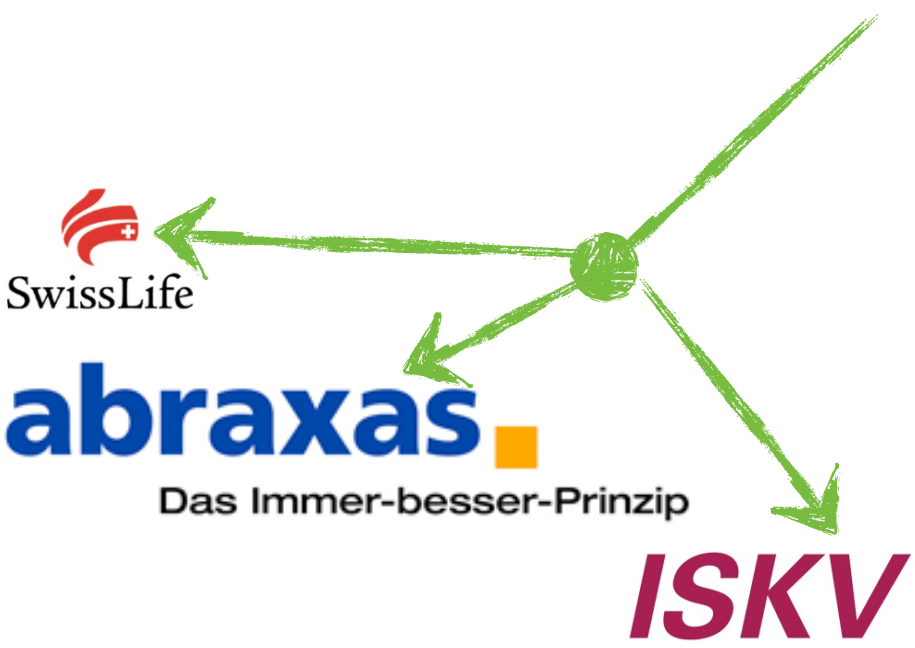
Joerg.Baechtiger@Abraxas.ch
http://www.xing.com/profile/Joerg_Baechtiger

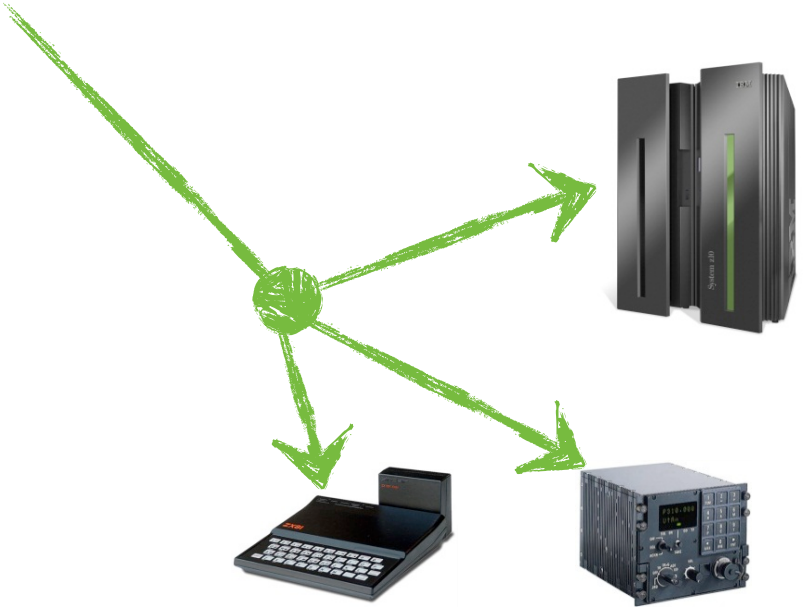

Übersicht geben
Ziele
Zusammenhänge aufzeigen
Interesse für Design Prinzipien wecken


Microsoft
.net
C/C++
Java


UNIFIED
MODELING
LANGUAGE
MODEL DRIVEN
ARCHITECTURE

Über mich

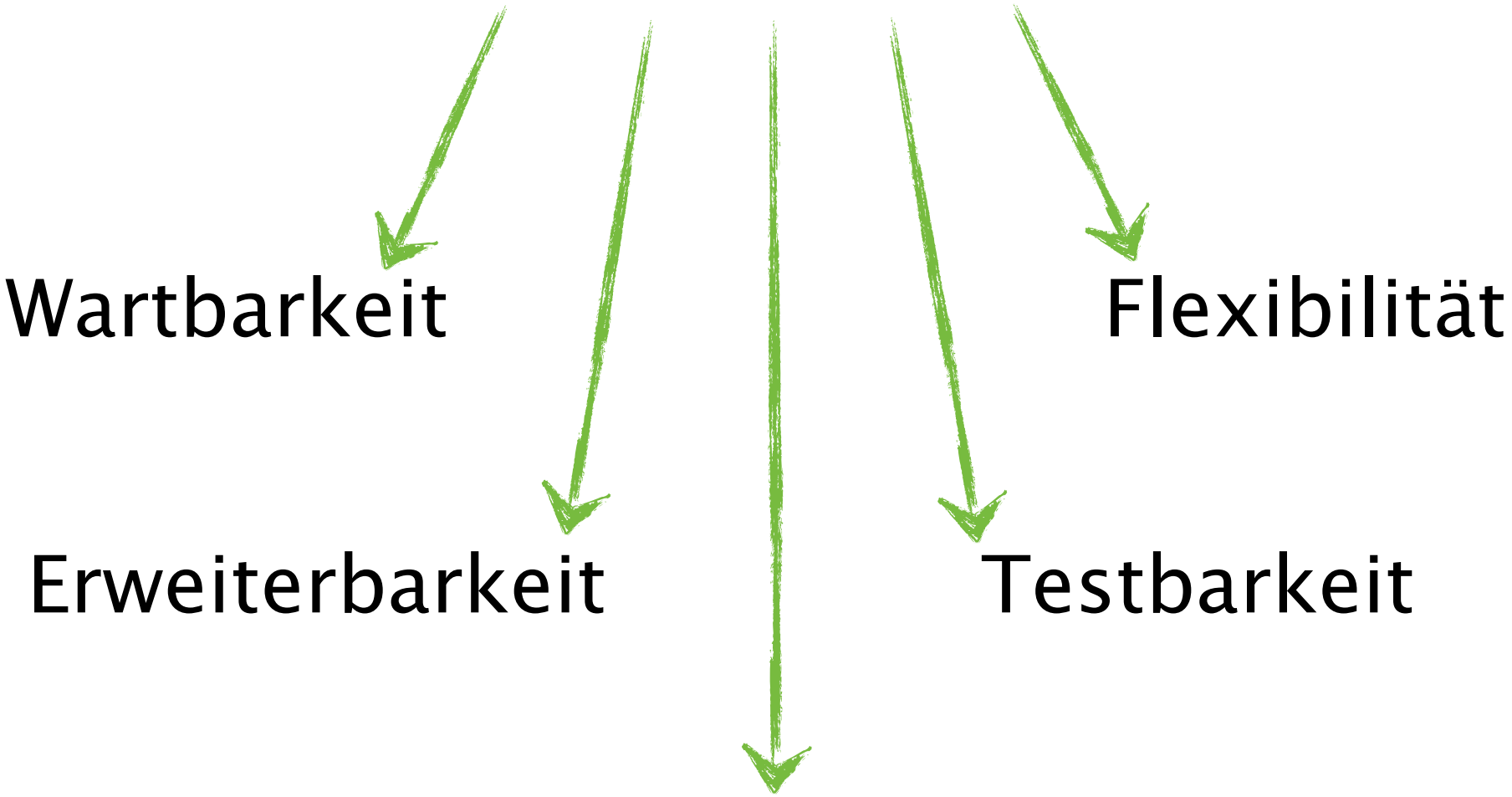

SwissLife
abraxas
Das Immer-besser-Prinzip
ISKV



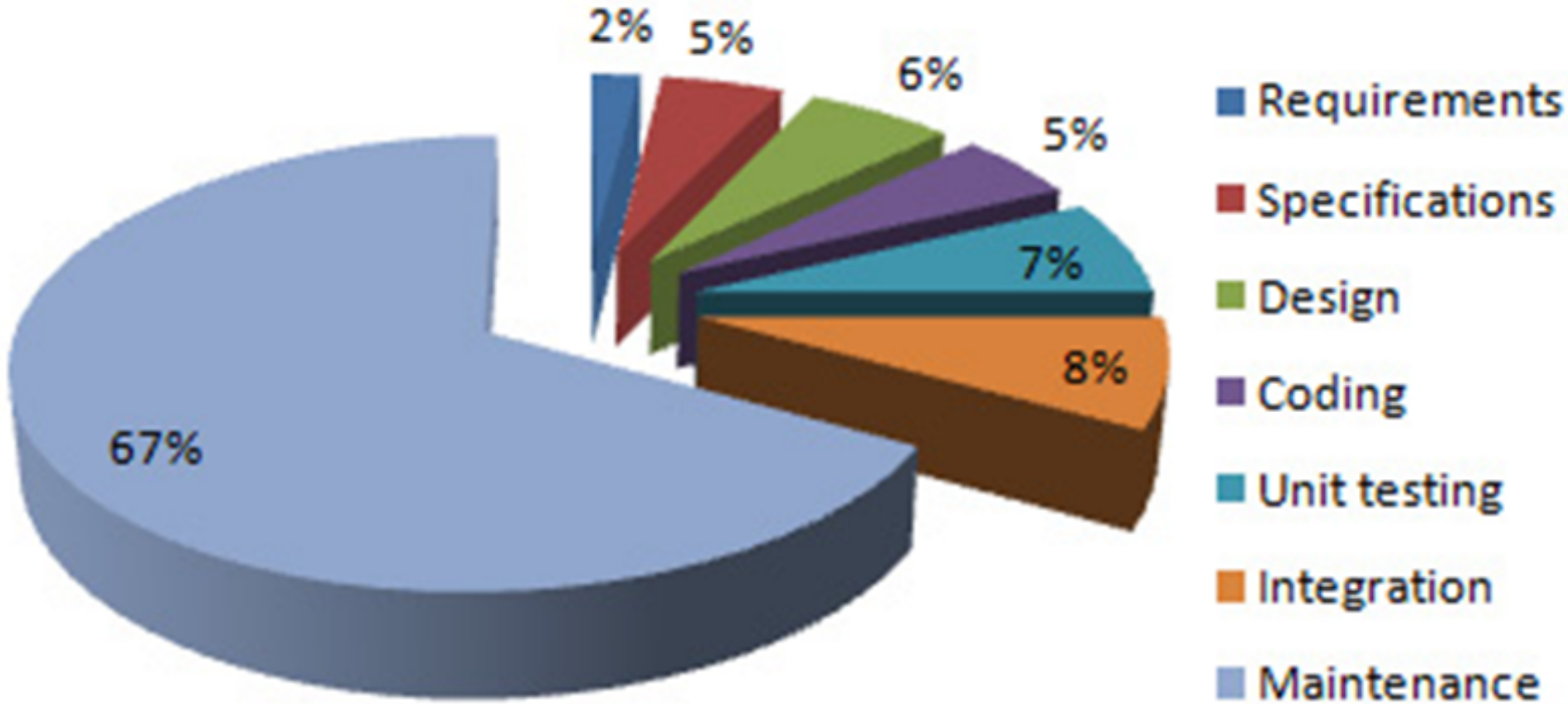
Die Macht, die uns umgibt
Design Principles.
Schneller und besser Software entwickeln

Schneller und besser Software entwickeln

Schneller und besser Software entwickeln

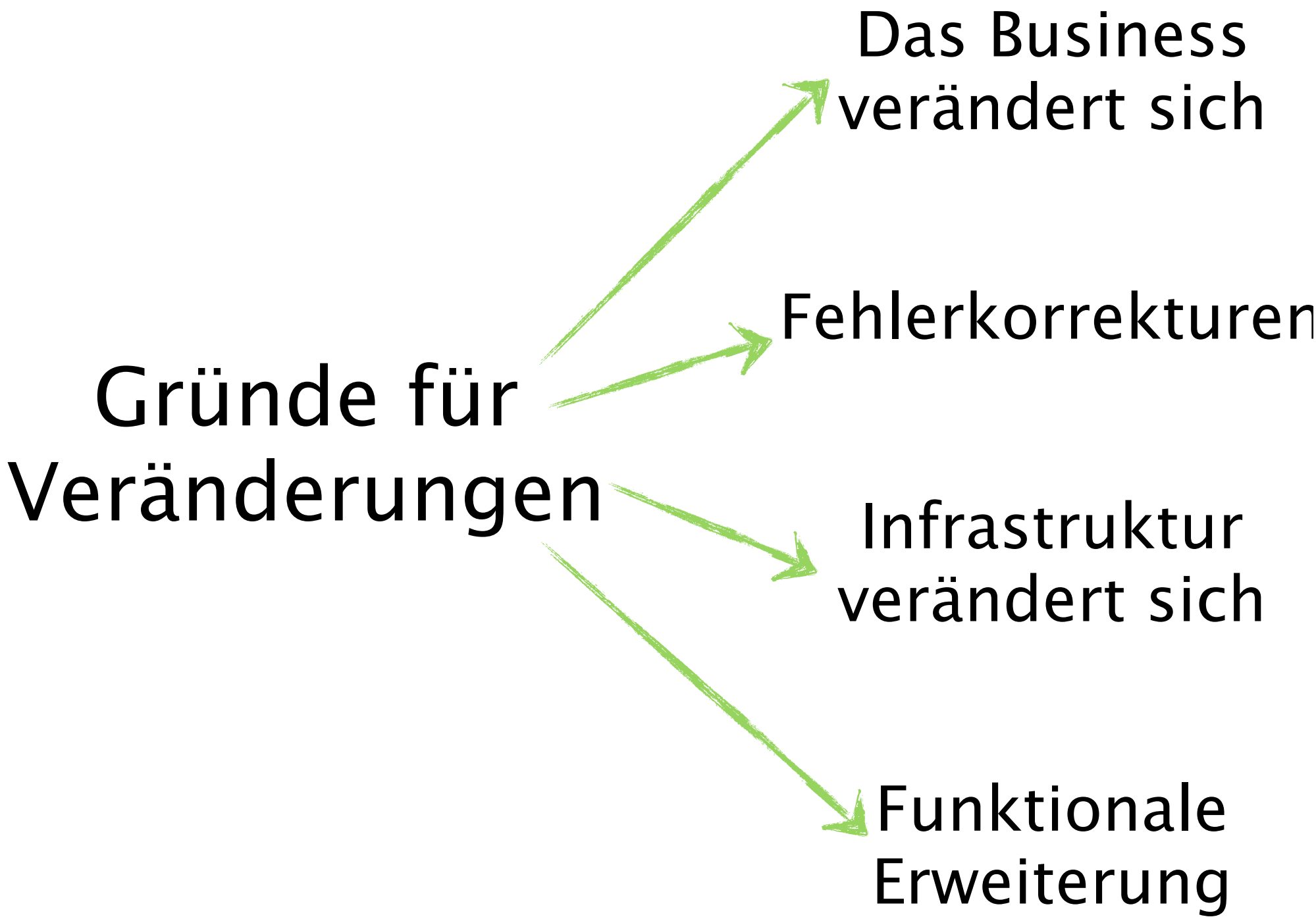


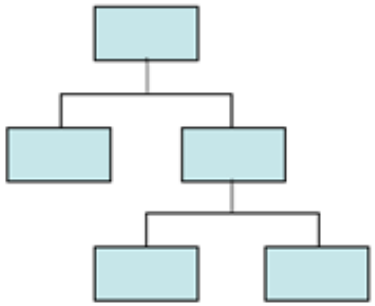
Software Life-Cycle Costs



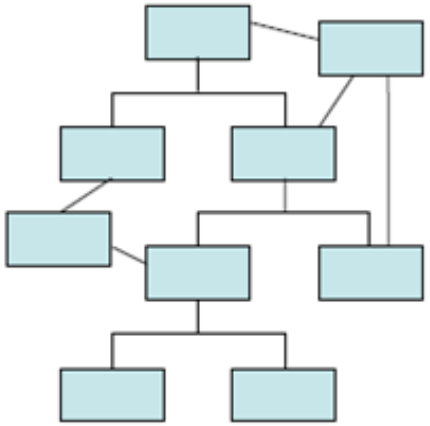
Source : Digital Aggregates

Year	Proportion of software maintenance costs	Definition	Reference
2000	**>90%**	Software cost devoted to system maintenance & evolution / total software costs	Erlikh (2000)
1993	**75%**	Software maintenance / information system budget (in Fortune 1000 companies)	Eastwood (1993)
1990	**>90%**	Software cost devoted to system maintenance & evolution / total software costs	Moad (1990)
1990	**60-70%**	Software maintenance / total management information systems (MIS) operating budgets	Huff (1990)
1988	**60-70%**	Software maintenance / total management information systems (MIS) operating budgets	Port (1988)
1984	**65-75%**	Effort spent on software maintenance / total available software engineering effort.	McKee (1984)
1981	**>50%**	Staff time spent on maintenance / total time (in 487 organizations)	Lientz & Swanson (1981)
1979	**67%**	Maintenance costs / total software costs	Zelkowitz *et al.* (1979)

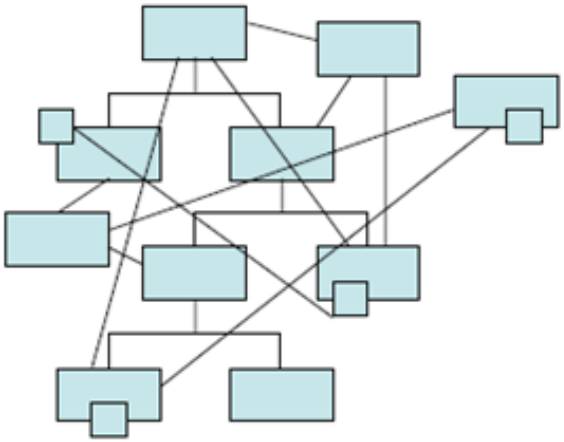

Gründe für Veränderungen
Das Business verändert sich
Fehlerkorrekturen
Infrastruktur verändert sich
Funktionale Erweiterung



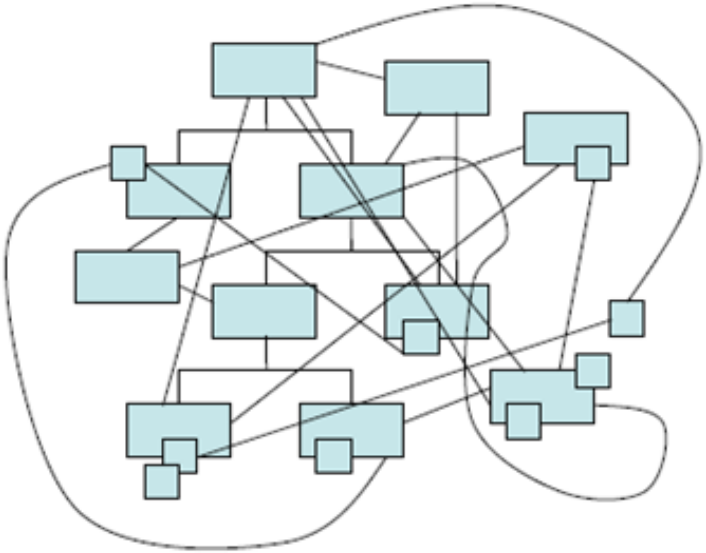
Cost of change: C



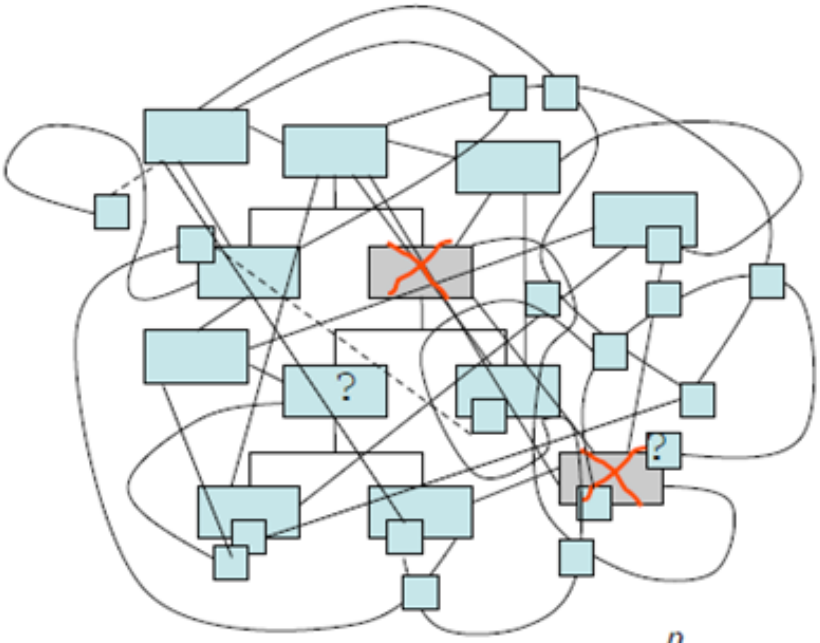
Cost of change: C + *n*



Cost of change: C x *n*

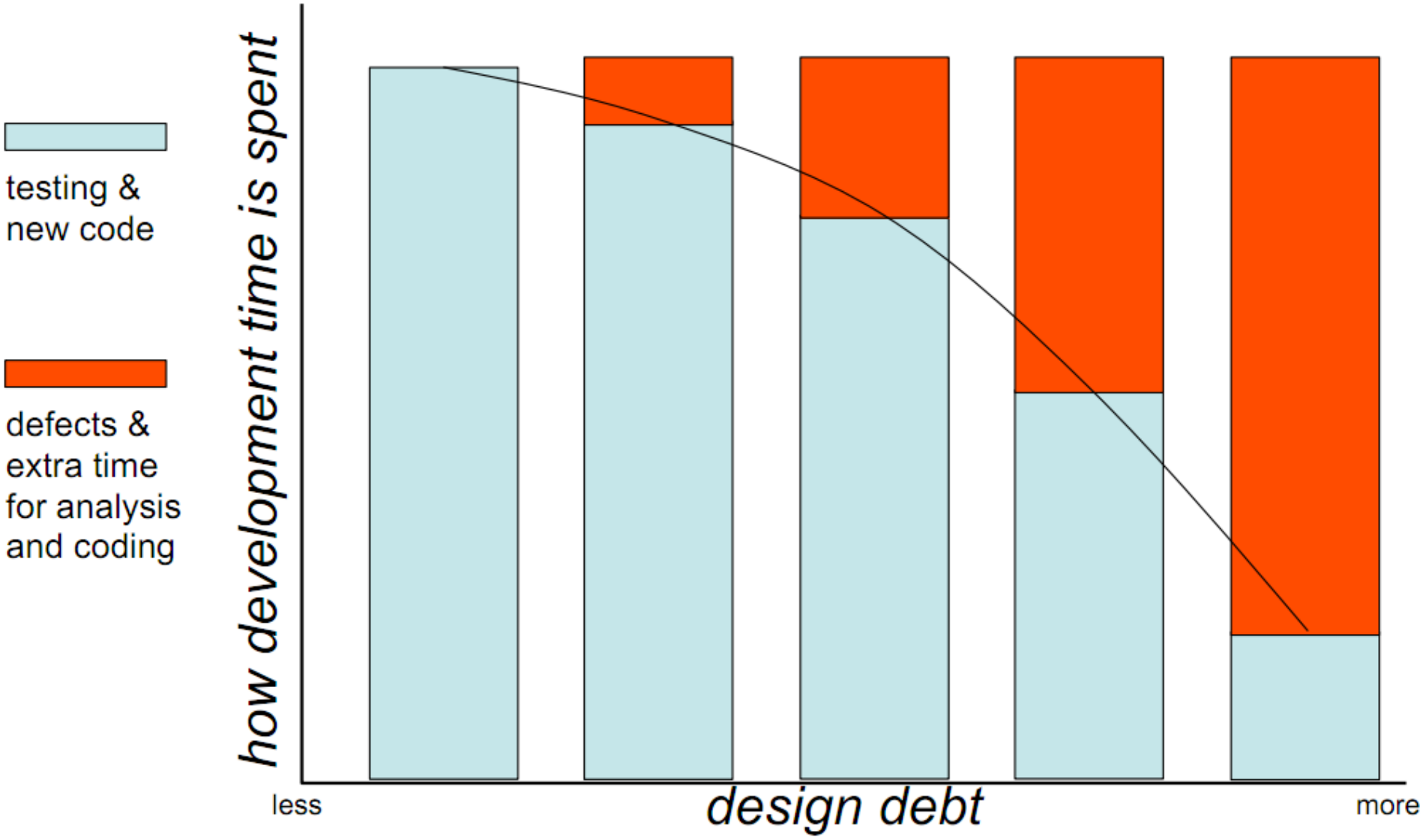


Cost of change: C^{n}



Cost of change: $C^{n^{n}}$

Slides by *Dave Nicolette*

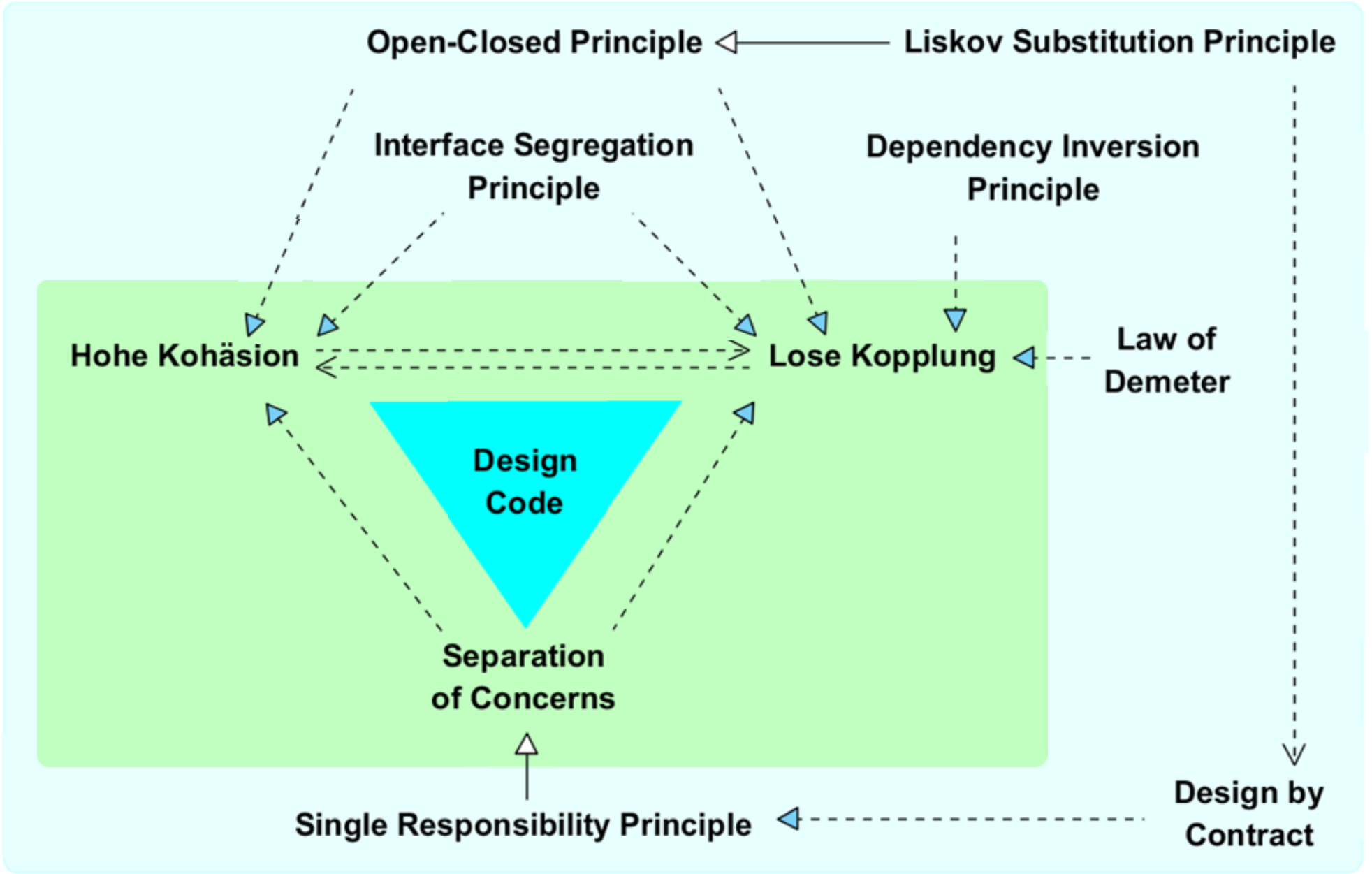


Slides by *Dave Nicolette*

Design Prinzipien

Schneller und besser Software entwickeln

Übersicht

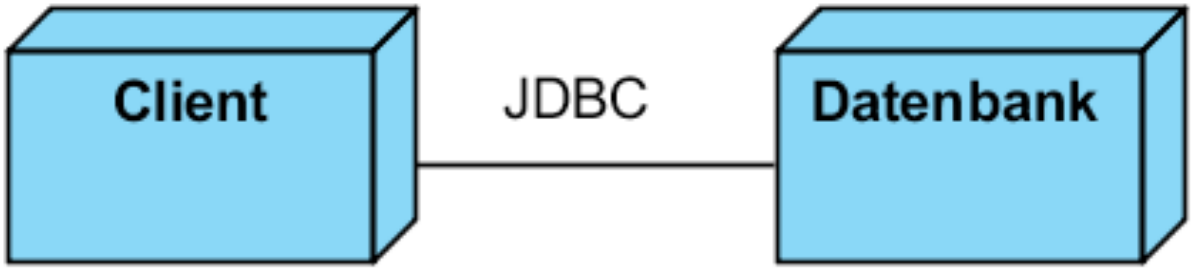

Open-Closed Principle
Liskov Substitution Principle
Interface Segregation Principle
Dependency Inversion Principle
Hohe Kohäsion
Lose Kopplung
Law of Demeter
Design Code
Separation of Concerns
Single Responsibility Principle
Design by Contract

Design Principles in Aktion

Informationen vom Auftraggeber:

- Verleih von DVDs
- Kunde bezahlt bei der Rückgabe der DVDs
- Der Preis ist abhängig von der Ausleihdauer
- Die Benutzer der Software sind die Angestellten im DVD Shop

DVD Verleih V1.0


Client
JDBC
Datenbank

Separation of Concerns

- Benutzer Interface
- Logik
- Datenhaltung
- Transaktion

HTML/CSS/HTTP

Vertragssystem

Partnersystem

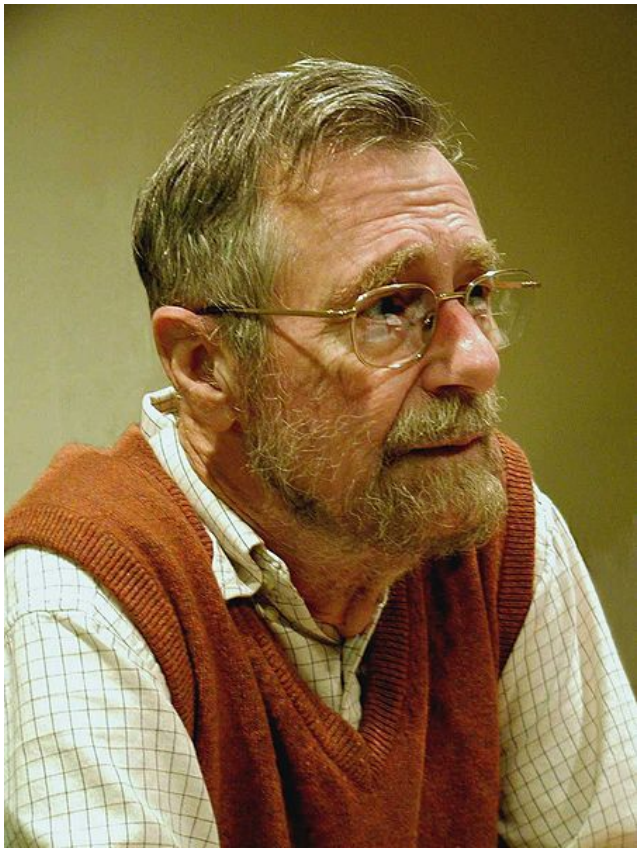
Arbeitsteilung in der Entwicklung

Application Server

AOP

Layers

MDA



Edsger W. Dijkstra

On the role of scientific thought, 1974

http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html

Separation of Concerns

Technisch
Fachlich
Infrastruktur
Horizontal
Vertikal

Zuständigkeit
Verantwortung

Konsequenzen

Entkopplung der Elemente

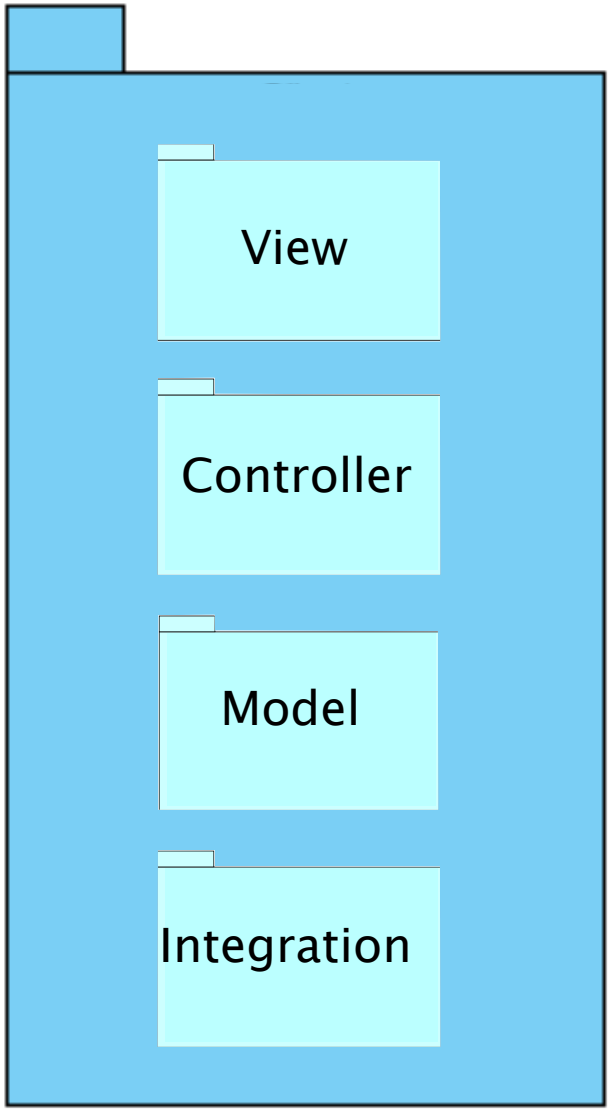
Reduziert (lokale) Komplexität

Time to market wird verkürzt
(parallele Entwicklung)

Verhindert Redundanz
und Inkonsistenz

Delegation führt zur Kopplung

Client



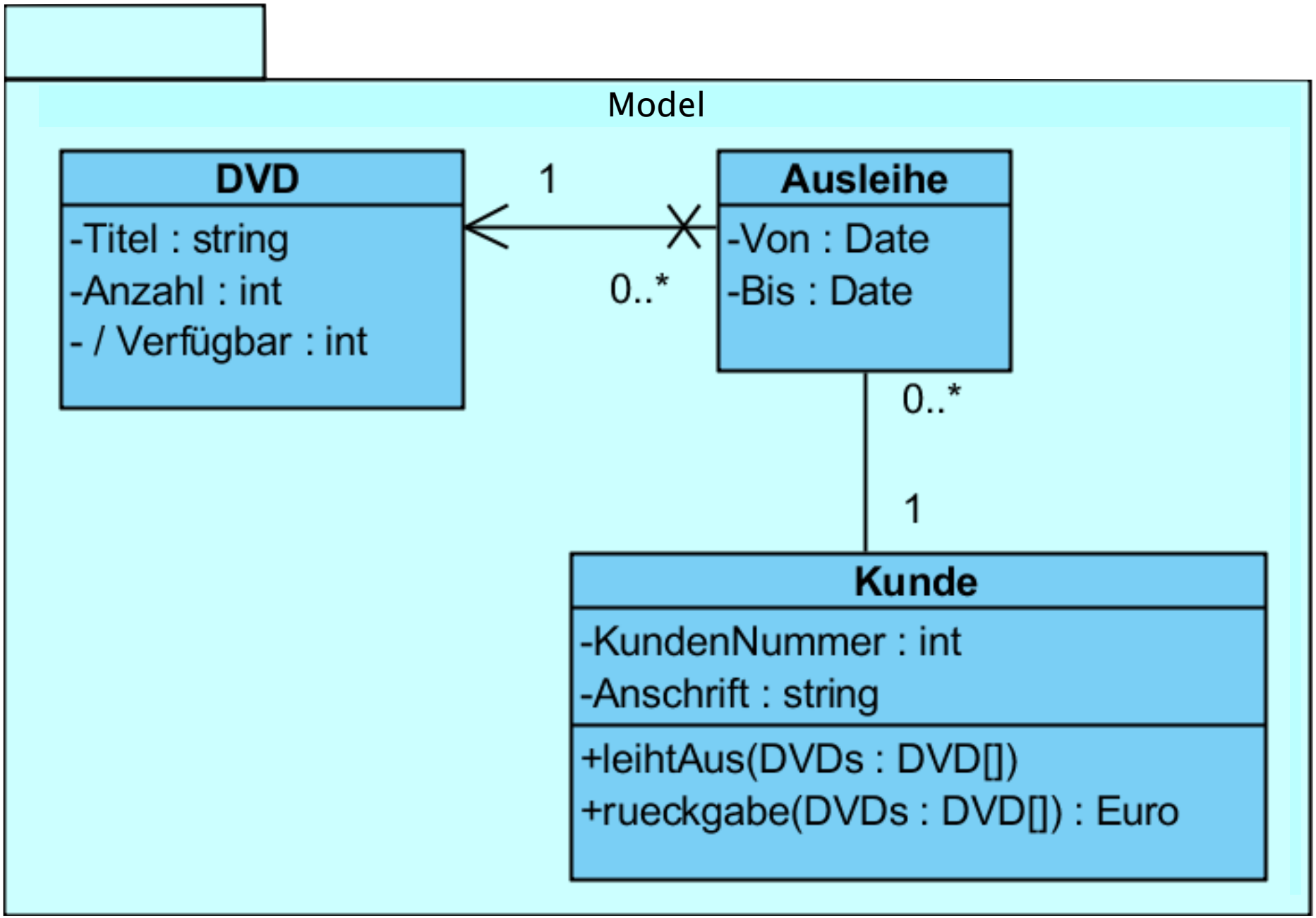
Concerns:

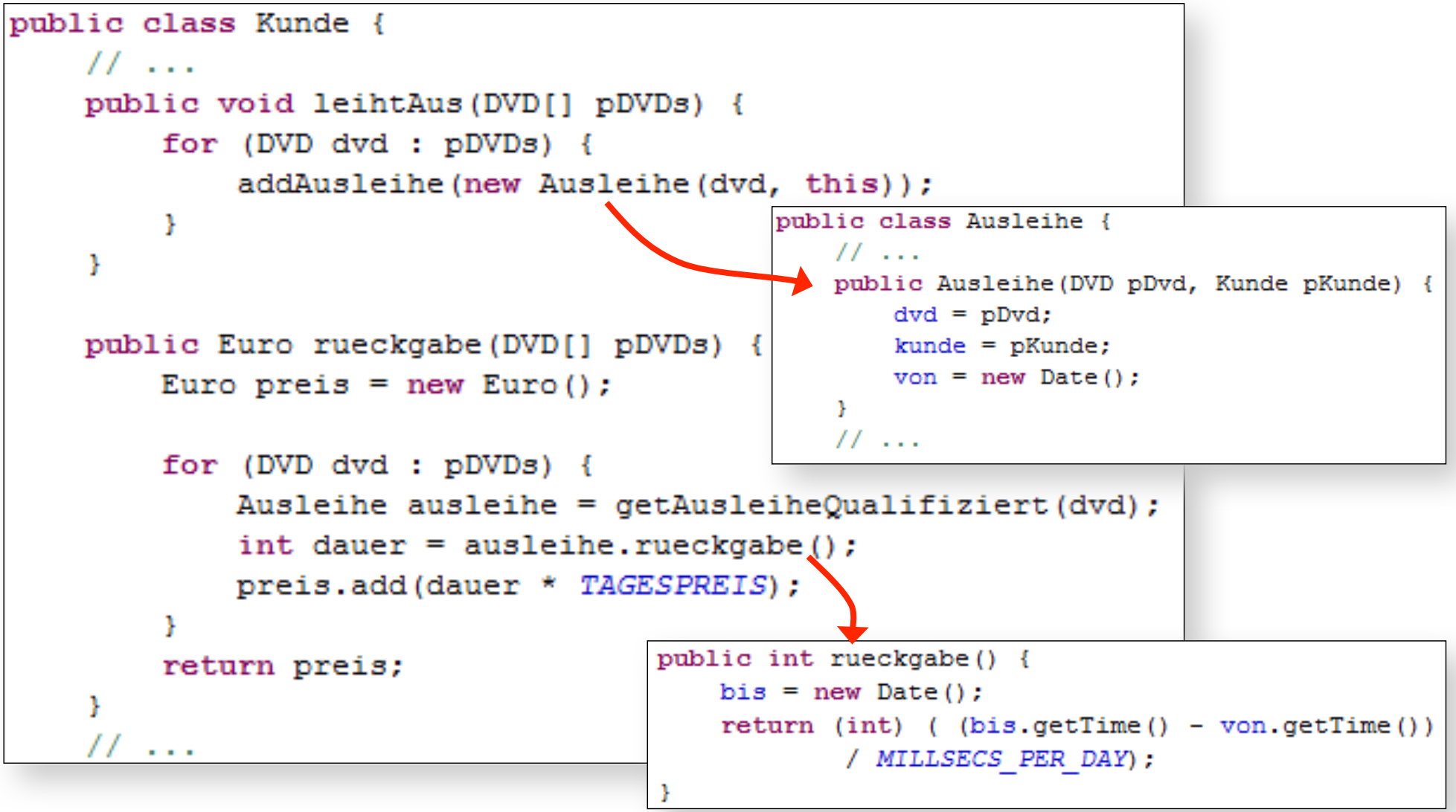
Hauptfenster
Dialoge

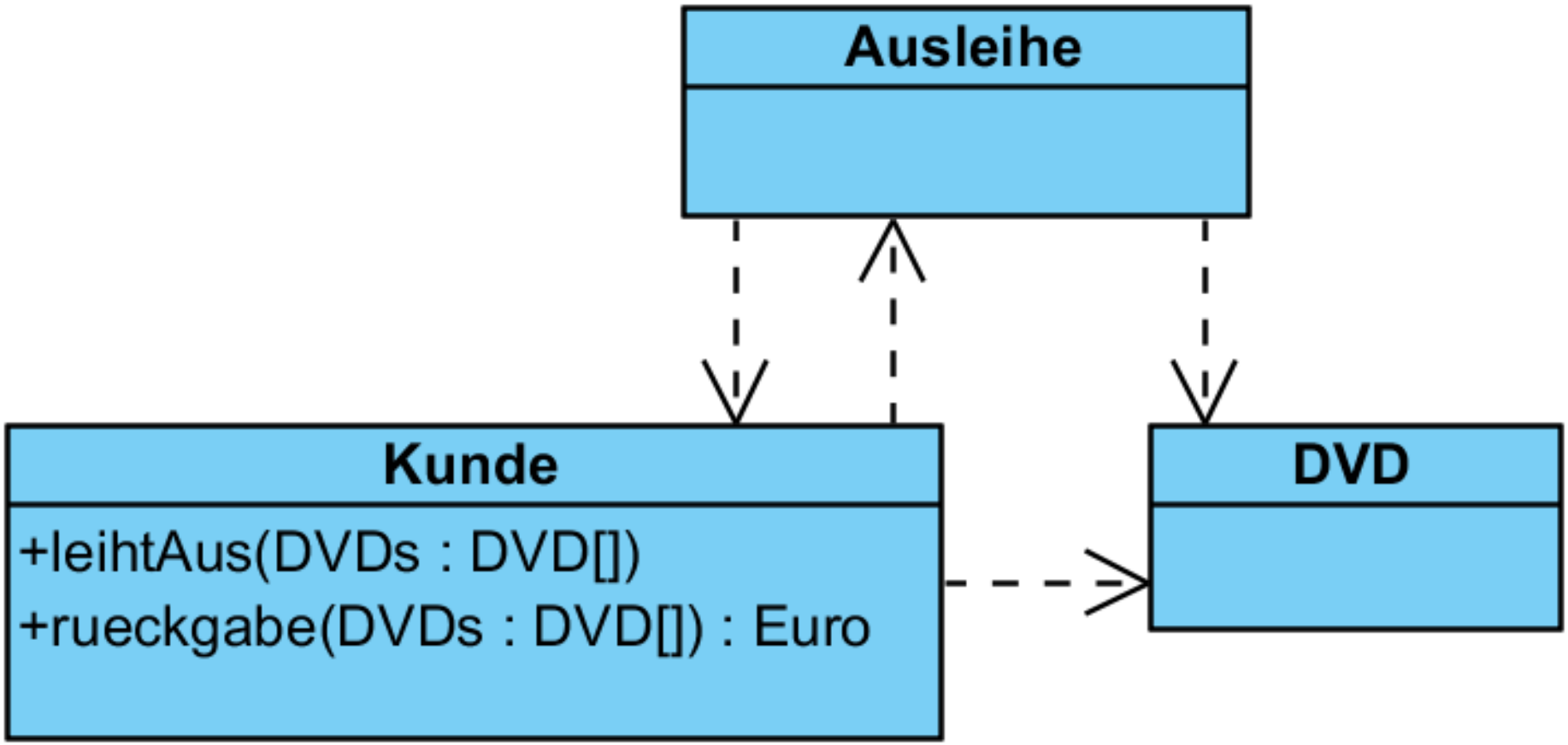
Interaktionen

Daten
Logik

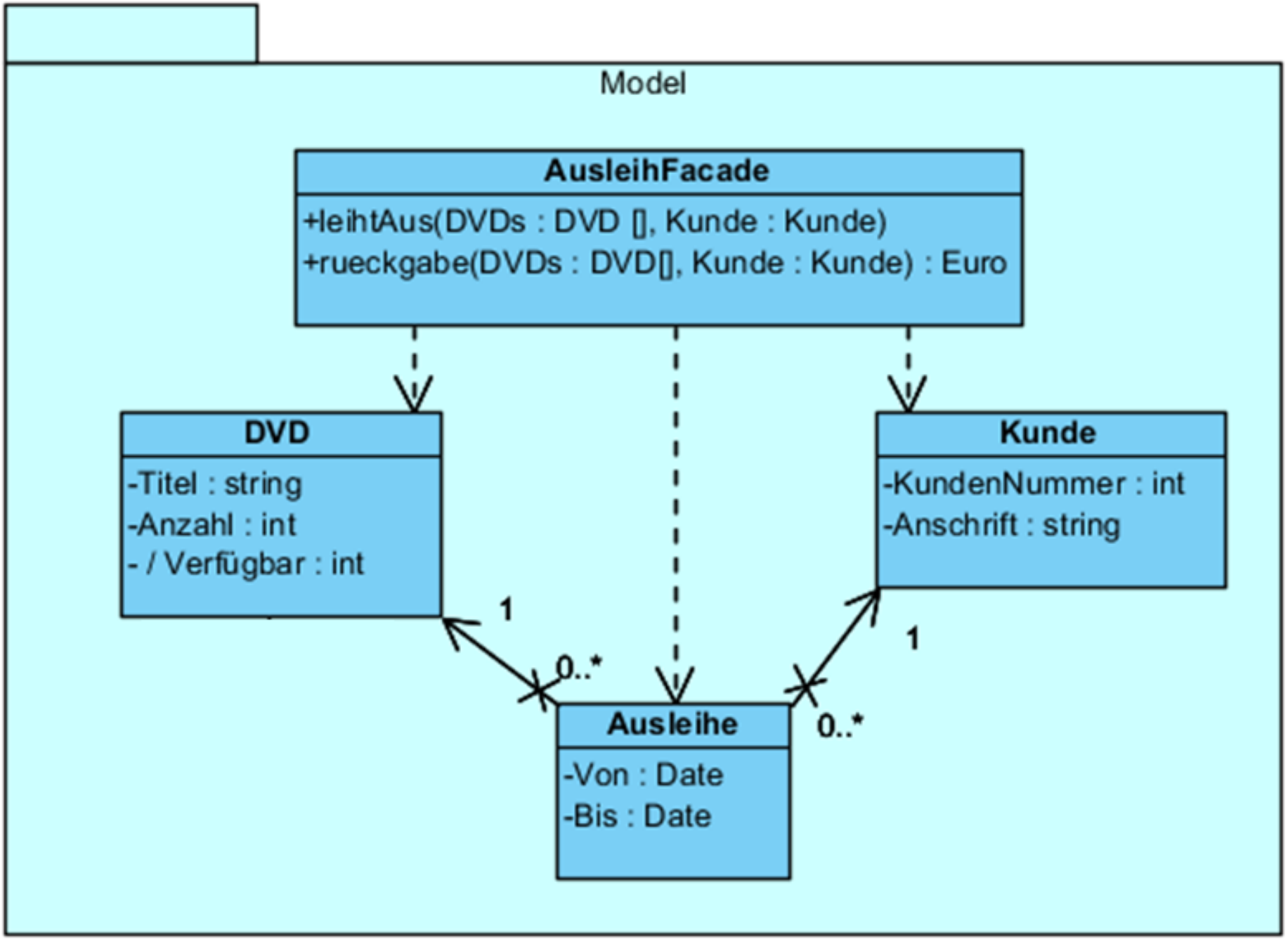
Zugriff auf ext. Systeme


Model
DVD
-Titel : string
-Anzahl : int
- / Verfügbar : int
1
0..*
Ausleihe
-Von : Date
-Bis : Date
0..*
1
Kunde
-KundenNummer : int
-Anschrift : string
+leihtAus(DVDs : DVD[])
+rueckgabe(DVDs : DVD[]) : Euro


public class Kunde {
// ...
public void leihtAus(DVD[] pDVDs) {
for (DVD dvd : pDVDs) {
addAusleihe(new Ausleihe(dvd, this));
}
}
public Euro rueckgabe(DVD[] pDVDs) {
Euro preis = new Euro();
for (DVD dvd : pDVDs) {
Ausleihe ausleihe = getAusleiheQualifiziert(dvd);
int dauer = ausleihe.rueckgabe();
preis.add(dauer * TAGESPREIS);
}
return preis;
}
// ...
public class Ausleihe {
// ...
public Ausleihe(DVD pDvd, Kunde pKunde) {
dvd = pDvd;
kunde = pKunde;
von = new Date();
}
// ...
public int rueckgabe() {
bis = new Date();
return (int) ((bis.getTime() - von.getTime())
/ MILLSECS_PER_DAY);
}


Ausleihe
Kunde
+leihtAus(DVDs : DVD[])
+rueckgabe(DVDs : DVD[]) : Euro
DVD

DVD Verleih V1.1


Model
AusleihFacade
+leihtAus(DVDs : DVD [], Kunde : Kunde)
+rueckgabe(DVDs : DVD[], Kunde : Kunde) : Euro
DVD
-Titel : string
-Anzahl : int
- / Verfügbar : int
Kunde
-KundenNummer : int
-Anschrift : string
Ausleihe
-Von : Date
-Bis : Date
1
0..*
1
0..*

```java
public class AusleihFacade {

    // ...
    public void leihtAus(DVD[] pDVDs, Kunde pKunde) {
        for (DVD dvd : pDVDs) {
            store(new Ausleihe(dvd, pKunde));
        }
    }

    public Euro rueckgabe(DVD[] pDVDs, Kunde pKunde) {
        Euro preis = new Euro();

        for (DVD dvd : pDVDs) {
            Ausleihe ausleihe = getAusleiheQualifiziert(dvd, pKunde);
            int dauer = ausleihe.rueckgabe();
            preis.add(dauer * TAGESPREIS);
        }
        return preis;
    }

    // ...
```

Single Responsibility Principle

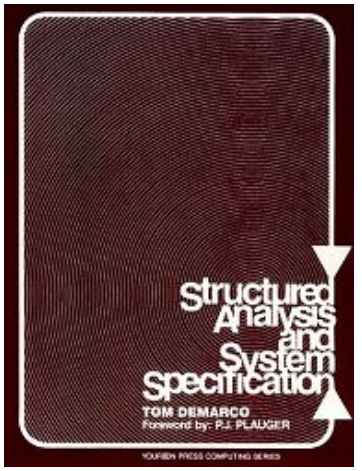
Es sollte nie mehr als einen Grund dafür geben, ein Element zu ändern.

(ROBERT C. MARTIN: Agile Software Development: Principles, Patterns, and Practices)

Tom DeMarco

Structured Analysis and System Specification, 1979

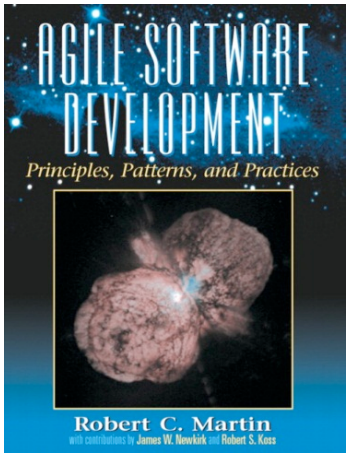
ISBN 0138543801



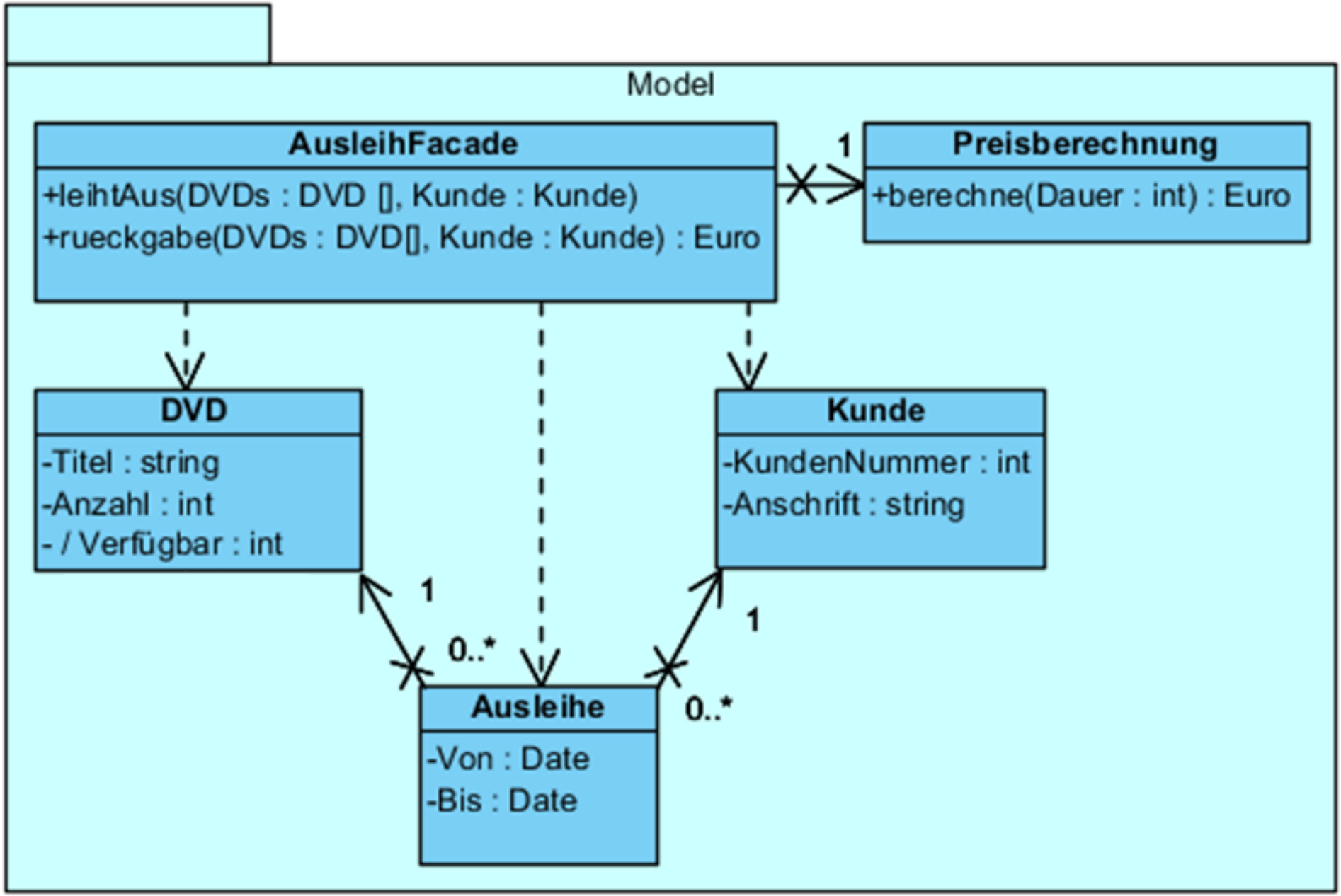
Robert C. Martin

Agile Software Development, Principles, Patterns, and Practice, 2003

ISBN 0135974445



DVD Verleih V1.2


Model
AusleihFacade
+leihtAus(DVDs : DVD [], Kunde : Kunde)
+rueckgabe(DVDs : DVD[], Kunde : Kunde) : Euro
1
Preisberechnung
+berechne(Dauer : int) : Euro
DVD
-Titel : string
-Anzahl : int
- / Verfügbar : int
Kunde
-KundenNummer : int
-Anschrift : string
1
0..*
1
0..*
Ausleihe
-Von : Date
-Bis : Date

```java
public class AusleihFacade {

    // ...
    public void leihtAus(DVD[] pDVDs, Kunde pKunde) {
        for (DVD dvd : pDVDs) {
            store(new Ausleihe(dvd, pKunde));
        }
    }

    public Euro rueckgabe(DVD[] pDVDs, Kunde pKunde) {
        Euro preis = new Euro();

        for (DVD dvd : pDVDs) {
            Ausleihe ausleihe = getAusleiheQualifiziert(dvd, pKunde);
            int dauer = ausleihe.rueckgabe();
            preis.add(preisRechner.berechne(dauer));
        }
        return preis;
    }
    // ...
```

```java
public class Preisberechnung {

    // ...

    public Euro berechne(int pDauer) {
        return new Euro(pDauer*TAGESPREIS);
    }

    // ...
```

Konsequenzen

Kleinere, fokussierte Elemente
mit einer tieferen Komplexität
und besserer Lesbarkeit

Höhere Kohäsion und losere Kopplung

Klare, eindeutige Namen

Grössere Flexibilität

Erhöhte Verwaltung durch eine
grössere Anzahl von Elementen

Bug-Report

```
java.lang.NullPointerException
at ch.abraxas.dvdShop.model.AusleihFacade.getAusleiheQualifiziert(AusleihFacade.java:31)
at ch.abraxas.dvdShop.model.AusleihFacade.rueckgabe(AusleihFacade.java:23)
at ch.abraxas.dvdShop.controller.Ausleihe.rueckgabe(Ausleihe.java:26)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
at java.lang.reflect.Method.invoke(Method.java:597)
```

```
29
30      private Ausleihe getAusleiheQualifiziert(DVD pDvd, Kunde pKunde) {
31          int kundenNummer = pKunde.getNummer();
```

```
public Euro rueckgabe(DVD[] pDVDs, Kunde pKunde) {
    Euro preis = new Euro();

    for (DVD dvd : pDVDs) {
        Ausleihe ausleihe = getAusleiheQualifiziert(dvd, pKunde);
        int dauer = ausleihe.rueckgabe();
        preis.add(preisRechner.berechne(dauer));
    }
    return preis;
}
```

DVD Verleih V1.2.1

```
public Euro rueckgabe(DVD[] pDVDs, Kunde pKunde) {
    Euro preis = new Euro();

    if ( (pDVDs != null) && (pDVDs.length > 0)
            && (pKunde != null) && (pKunde.getNummer() != 0)) {
        for (DVD dvd : pDVDs) {
            Ausleihe ausleihe = getAusleiheQualifiziert(dvd, pKunde);
            int dauer = ausleihe.rueckgabe();
            preis.add(preisRechner.berechne(dauer));
        }
    }
    return preis;
}
```

- Symptom bekämpft
- Höhere Komplexität
- Performance reduziert (ok, nur wenig ☺)
- Mehr Testfälle notwendig

IF Statements sind böse!

Design by Contract

Wenn Du zusagst, die Methode **nur bei erfüllten Vor-Bedingungen** aufzurufen, sage ich im **Gegenzug** zu, einen Endzustand zu liefern, in dem die **Nach-Bedingungen** erfüllt sind.

(Bertrand Meyer, Objektorientierte Software-Entwicklung)

Bertrand Meyer

Object-Oriented Software Construction, 1997

ISBN 0-136-29155-4

	Pflicht (Obligation)	Nutzen
Kunde	Erfüllt Vorbedingung	Verlässt sich auf Nachbedingung
Anbieter	Erfüllt Nachbedingung	Verlässt sich auf Vorbedingung

Non-redundancy principle

Unter keinen Umständen sollte die Implementierung einer Operation die Vorbedingungen prüfen.

(Bertrand Meyer, Objektorientierte Software-Entwicklung)

Konsequenzen

Explizite Vor- und Nachbedingungen für die Benutzung einer Methode

Weniger Fehler

Optionale Laufzeitprüfung

Bessere Erweiterbarkeit und Wartbarkeit durch Redundanzfreiheit

Höhere Performance

Tiefere Komplexität

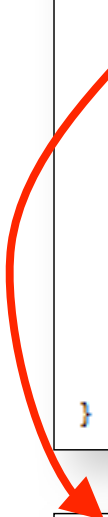
Implementierungen

Weder Java, C# und C++ unterstützen Design by Contract direkt in der Sprache.

Eine Liste von Tools ist auf dieser Site vorhanden:

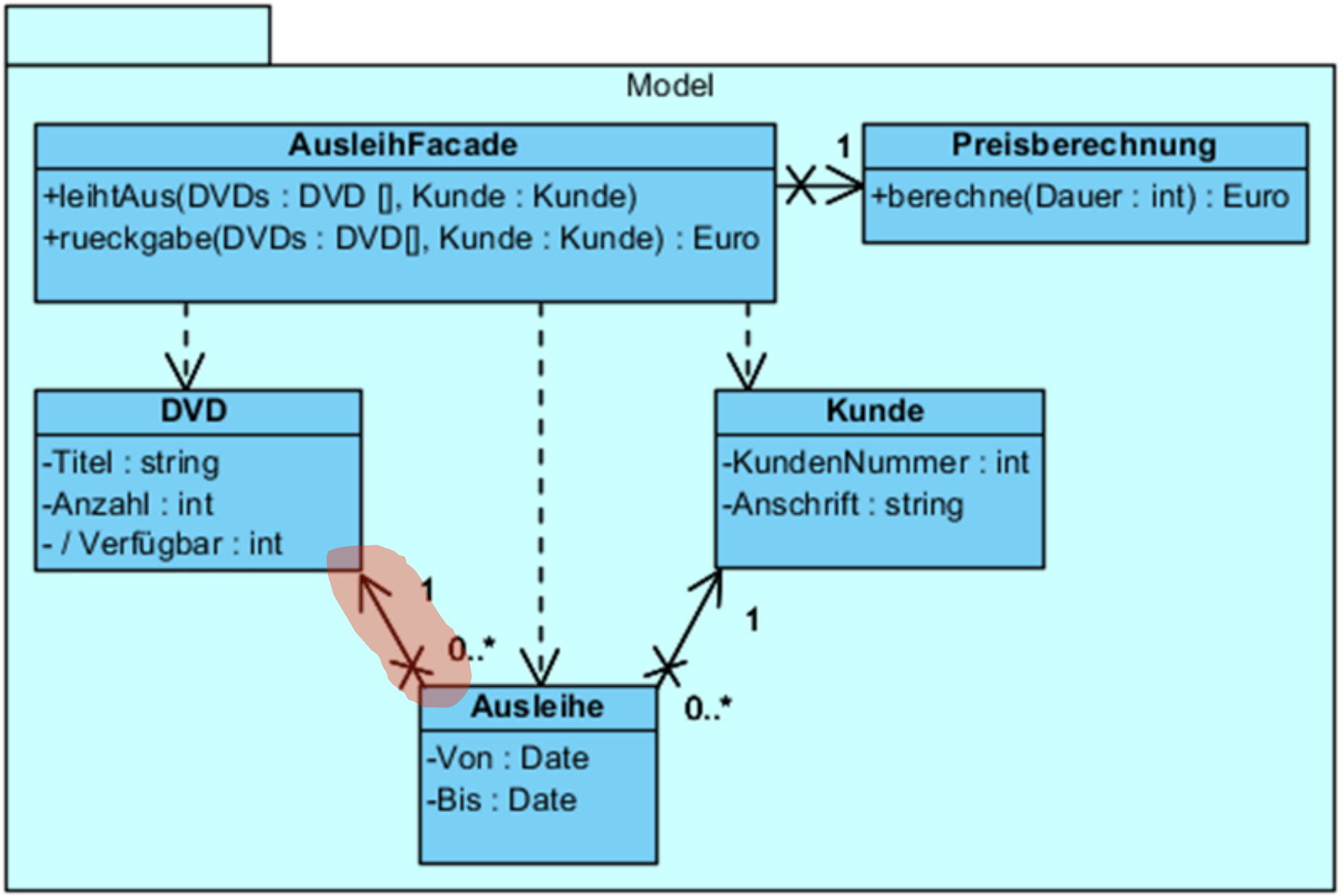
http://c2.com/cgi/wiki?DesignByContract

```java
public Euro rueckgabe(DVD[] pDVDs, Kunde pKunde) {
    Euro preis = new Euro();

    if ( (pDVDs != null) && (pDVDs.length > 0)
            && (pKunde != null) && (pKunde.getNummer() != 0)) {
        for (DVD dvd : pDVDs) {
            Ausleihe ausleihe = getAusleiheQualifiziert(dvd, pKunde);
            int dauer = ausleihe.rueckgabe();
            preis.add(preisRechner.berechne(dauer));
        }
    }
    return preis;
}
```

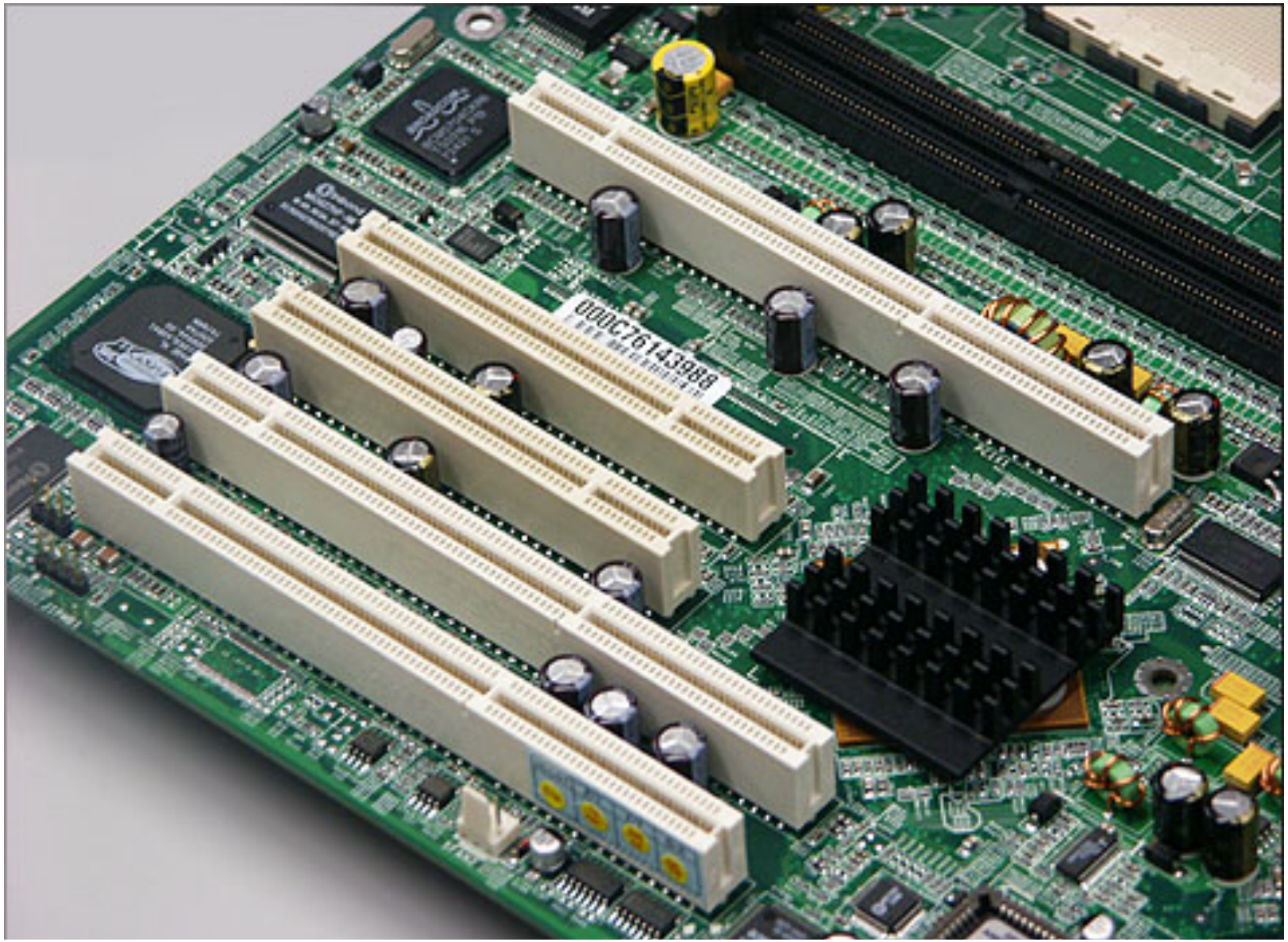


```java
@Requires({"pDVDs != null", "pDVDs.length > 0",
    "pKunde != null", "pKunde.getNummer() != 0"})
public Euro rueckgabe(DVD[] pDVDs, Kunde pKunde) {
    Euro preis = new Euro();

    for (DVD dvd : pDVDs) {
        Ausleihe ausleihe = getAusleiheQualifiziert(dvd, pKunde);
        int dauer = ausleihe.rueckgabe();
        preis.add(preisRechner.berechne(dauer));
    }

    return preis;
}
```


DVD
VIDEO GAMES
DVDs FOR SALE
DRAMA


Model
AusleihFacade
+leihtAus(DVDs : DVD [], Kunde : Kunde)
+rueckgabe(DVDs : DVD[], Kunde : Kunde) : Euro
1
Preisberechnung
+berechne(Dauer : int) : Euro
DVD
-Titel : string
-Anzahl : int
- / Verfügbar : int
Kunde
-KundenNummer : int
-Anschrift : string
1
0..*
1
0..*
Ausleihe
-Von : Date
-Bis : Date

Open-Closed Principle



Software sollte offen sein für Erweiterungen, aber geschlossen für Veränderungen

(ROBERT C. MARTIN: Agile Software Development: Principles, Patterns, and Practices)

Bertrand Meyer

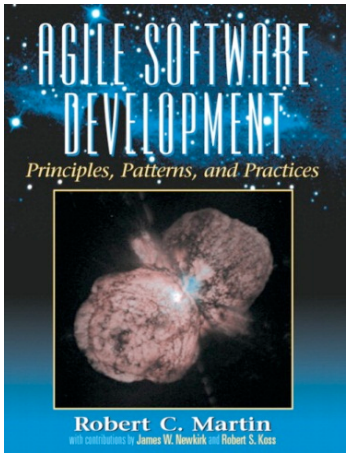
Object-Oriented Software Construction, 1997

ISBN 0-136-29155-4

Robert C. Martin

Agile Software Development, Principles, Patterns, and Practice, 2003

ISBN 0135974445



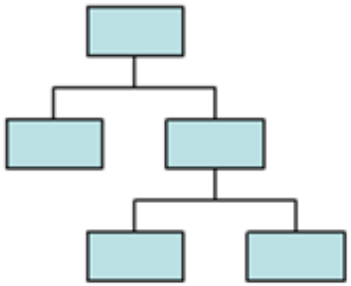
Offen für Erweiterungen

Geschlossen für Veränderungen

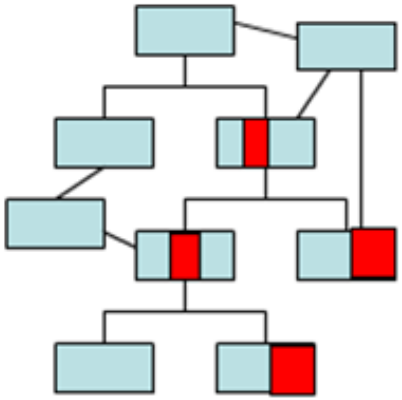
Abstraktion ist die Lösung

- Vererbung als Erweiterung
- Ersetzen von Bedingungen mit Hilfe von Polymorphismus

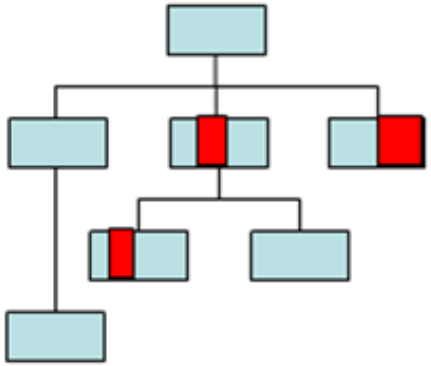
Üblich:



Starting code base

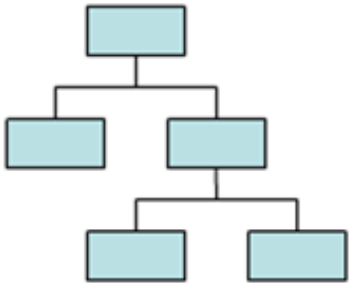


Changes implemented
red == code changed

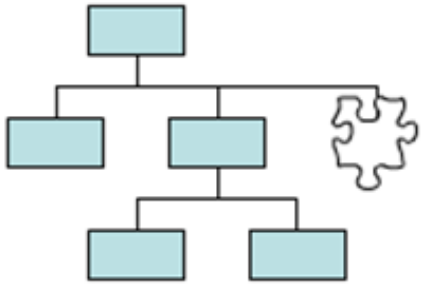


(Hopefully) Code cleaned up

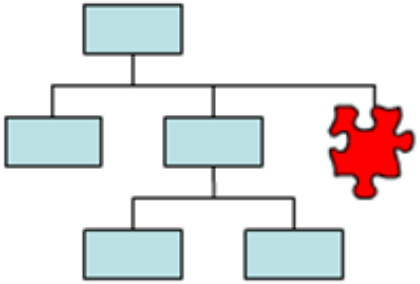
OCP:



Starting code base



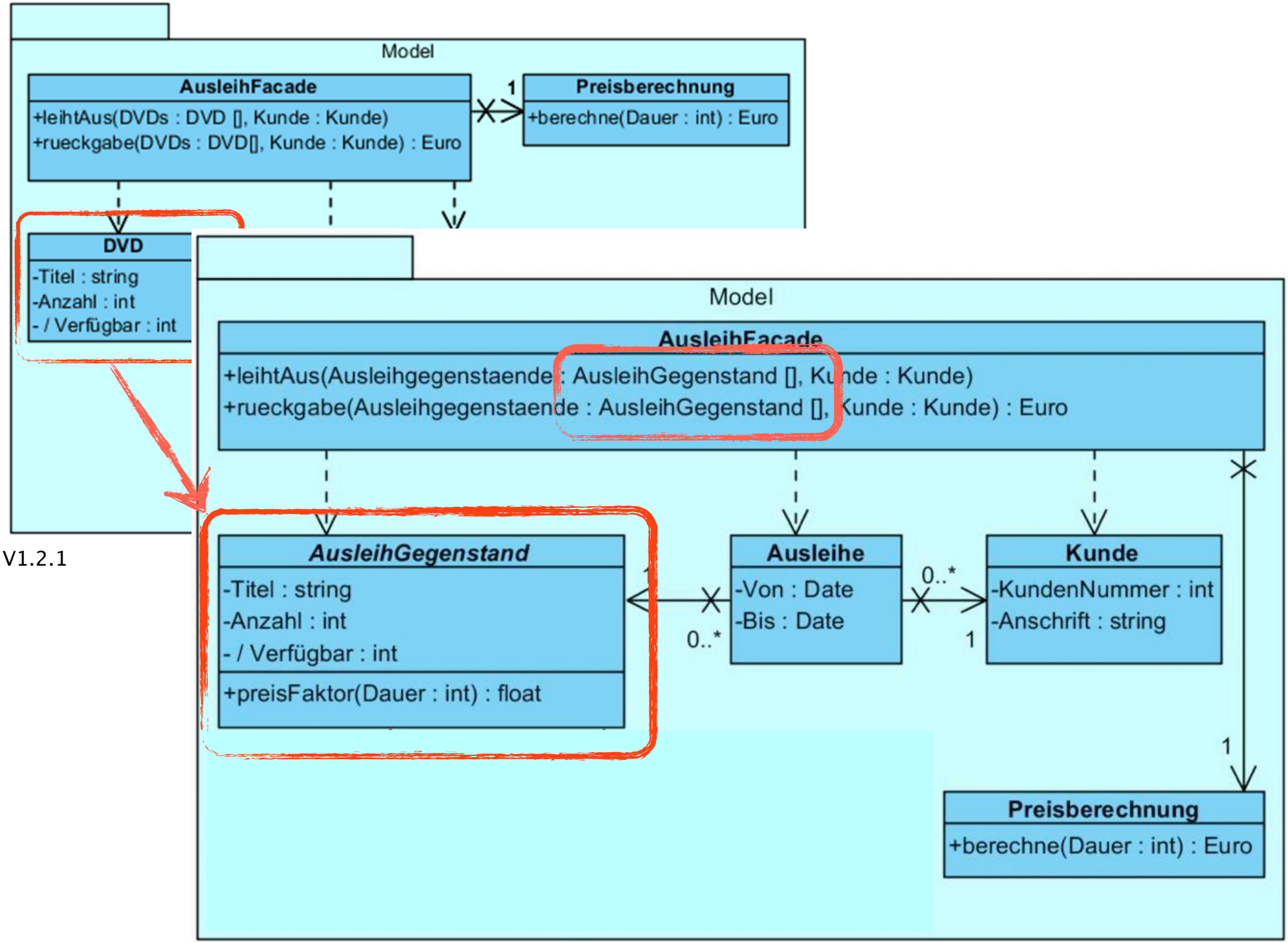
Change design to make
room for new feature



Implement feature

From a slide by Dave Nicolette

DVD/Game Verleih V2.0


Model
AusleihFacade
+leihtAus(DVDs : DVD [], Kunde : Kunde)
+rueckgabe(DVDs : DVD[], Kunde : Kunde) : Euro
1
Preisberechnung
+berechne(Dauer : int) : Euro
DVD
-Titel : string
-Anzahl : int
- / Verfügbar : int
Model
AusleihFacade
+leihtAus(Ausleihgegenstaende : AusleihGegenstand [], Kunde : Kunde)
+rueckgabe(Ausleihgegenstaende : AusleihGegenstand [], Kunde : Kunde) : Euro
AusleihGegenstand
-Titel : string
-Anzahl : int
- / Verfügbar : int
+preisFaktor(Dauer : int) : float
Ausleihe
-Von : Date
-Bis : Date
0..*
0..*
1
Kunde
-KundenNummer : int
-Anschrift : string
1
Preisberechnung
+berechne(Dauer : int) : Euro

V1.2.1

Konsequenzen

Erweiterbarkeit wird verbessert

Komplexität wird reduziert
(keine if, resp. instanceOfs)

Höhere Produktivität

Bessere Wiederverwendbarkeit

Flexibilität wird verbessert

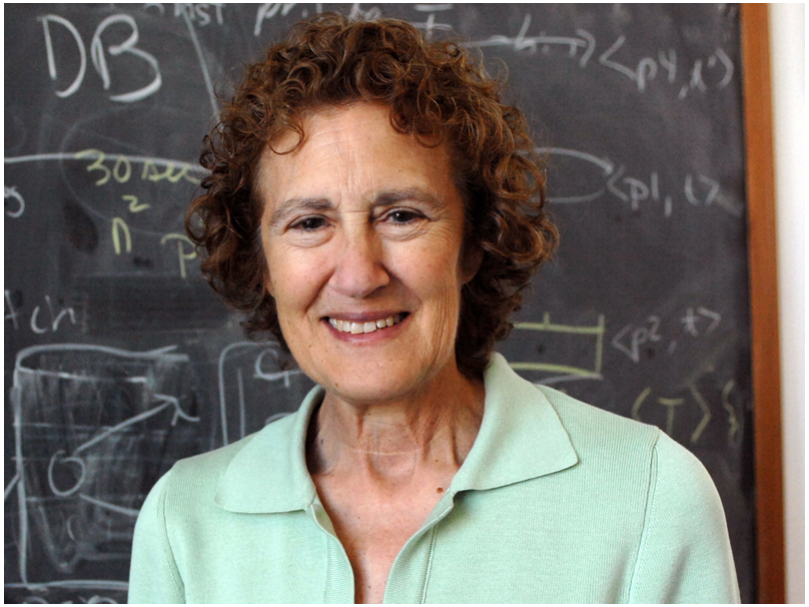
Liskov Substitution Principle

Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T. Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.

(Barbara H. Liskov, Jeannette M. Wing)

Ein Sub-Typ erfüllt den
Vertrag des Basis-Typs

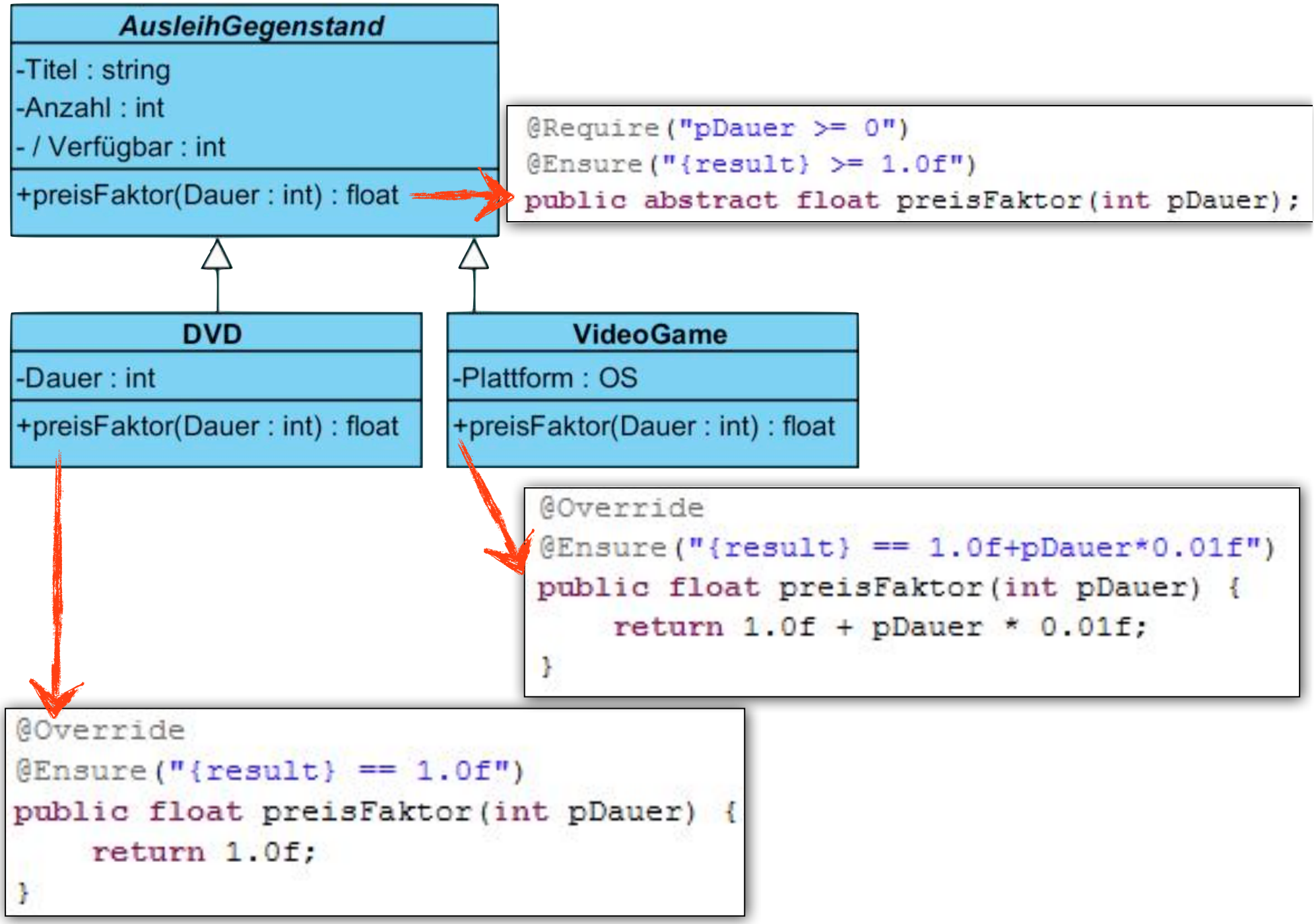
Ein Sub-Typ kann den
Basis-Typ ersetzen



Barbara Liskov

A Behavioral Notion of Subtyping
Publikation 1993

http://www.cse.ohio-state.edu/~neelam/courses/788/lwb.pdf

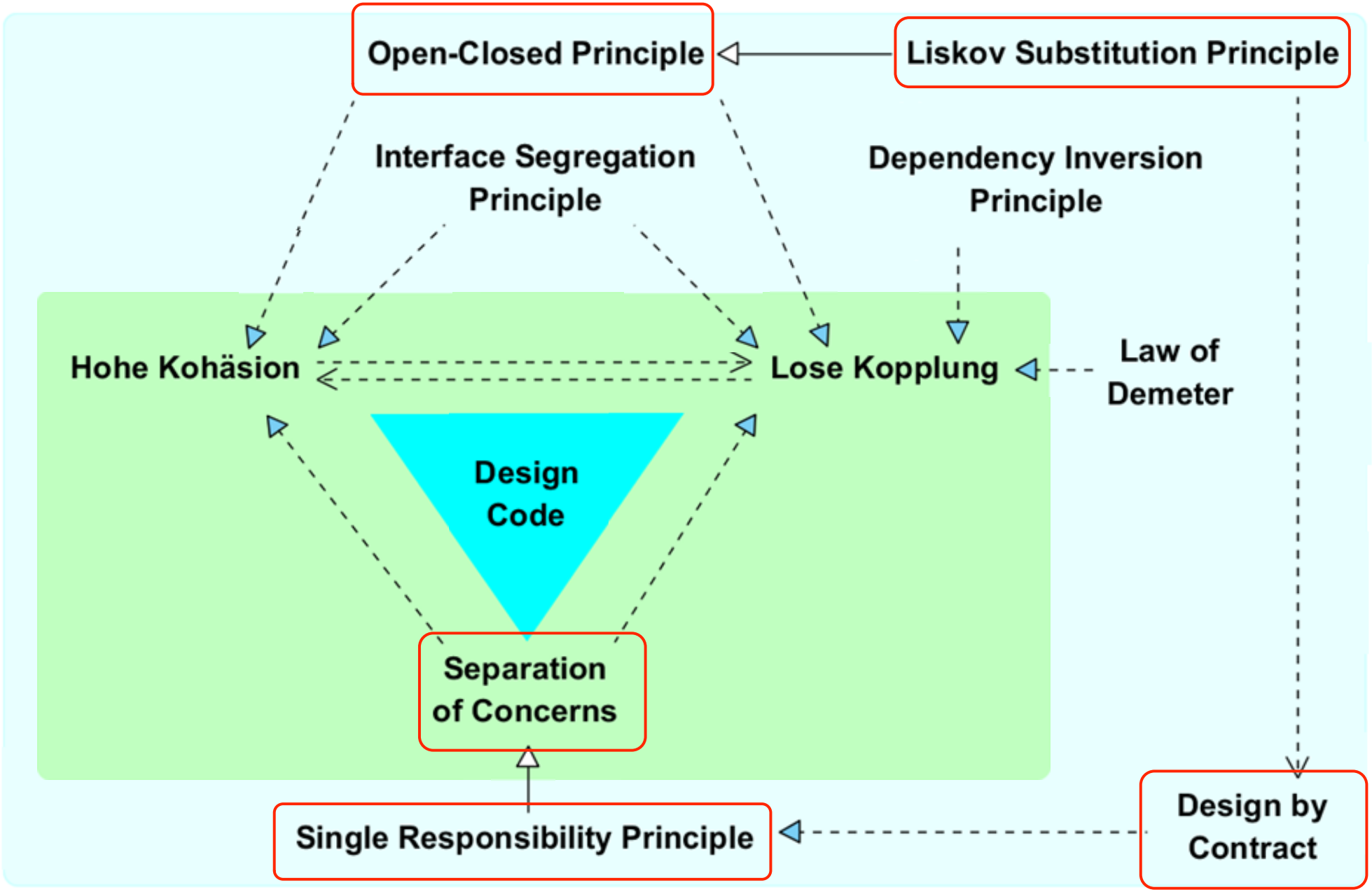

AusleihGegenstand
-Titel : string
-Anzahl : int
- / Verfügbar : int
+preisFaktor(Dauer : int) : float
@Require("pDauer >= 0")
@Ensure("{result} >= 1.0f")
public abstract float preisFaktor(int pDauer);
DVD
-Dauer : int
+preisFaktor(Dauer : int) : float
VideoGame
-Plattform : OS
+preisFaktor(Dauer : int) : float
@Override
@Ensure("{result} == 1.0f+pDauer*0.01f")
public float preisFaktor(int pDauer) {
return 1.0f + pDauer * 0.01f;
}
@Override
@Ensure("{result} == 1.0f")
public float preisFaktor(int pDauer) {
return 1.0f;
}

Pre-Conditions nicht strenger machen, aber weiter

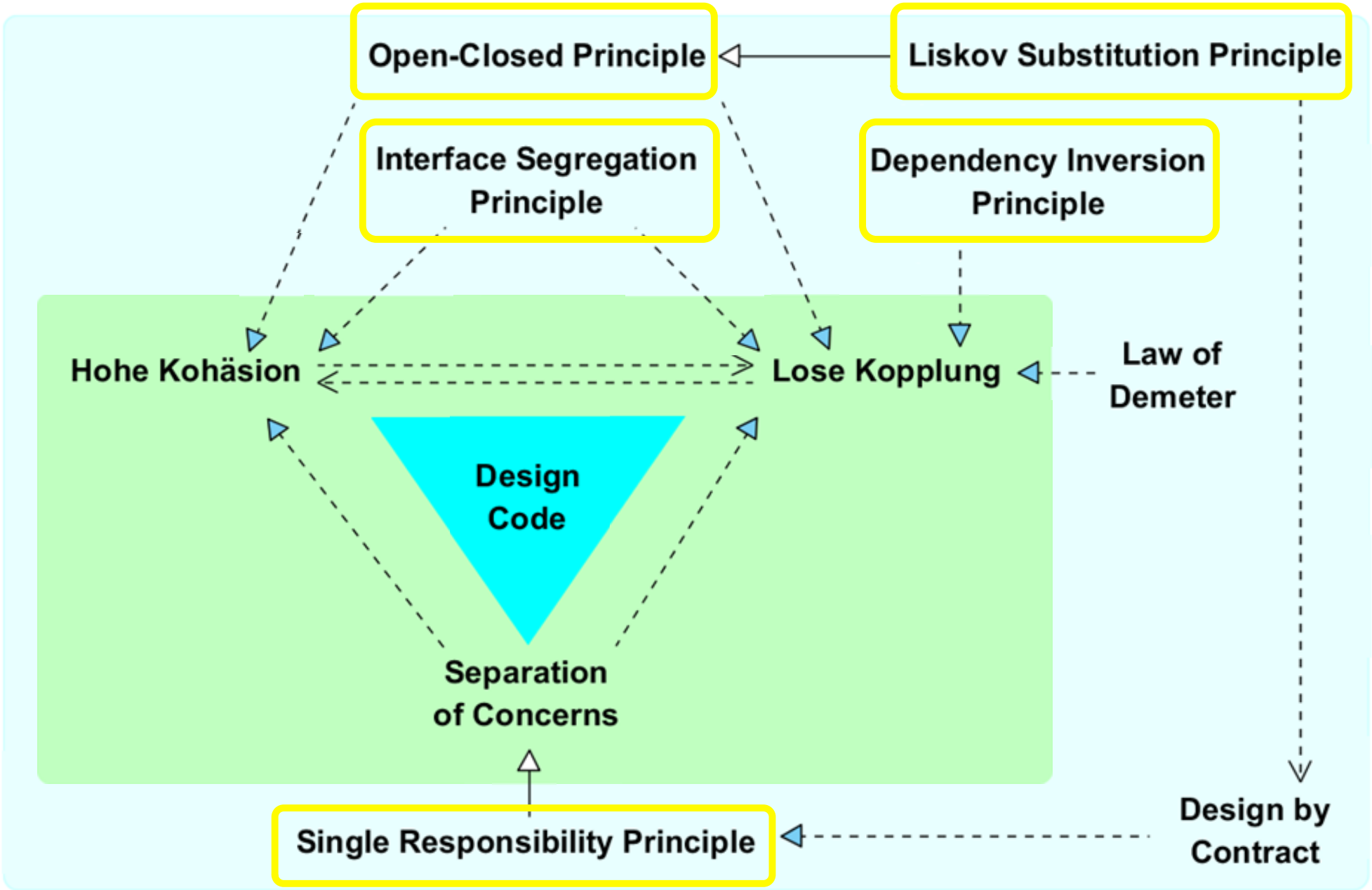
Post-Conditions nicht weiter machen, aber enger

Invarianten nicht verändern

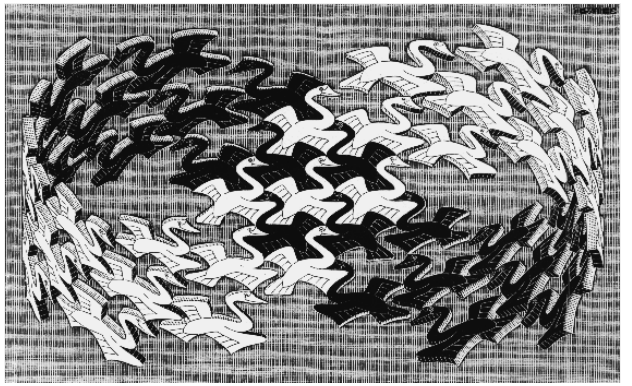
Übersicht


Open-Closed Principle
Liskov Substitution Principle
Interface Segregation Principle
Dependency Inversion Principle
Hohe Kohäsion
Lose Kopplung
Law of Demeter
Design Code
Separation of Concerns
Single Responsibility Principle
Design by Contract

Übersicht


Open-Closed Principle
Liskov Substitution Principle
Interface Segregation Principle
Dependency Inversion Principle
Hohe Kohäsion
Lose Kopplung
Law of Demeter
Design Code
Separation of Concerns
Single Responsibility Principle
Design by Contract

Design Patterns

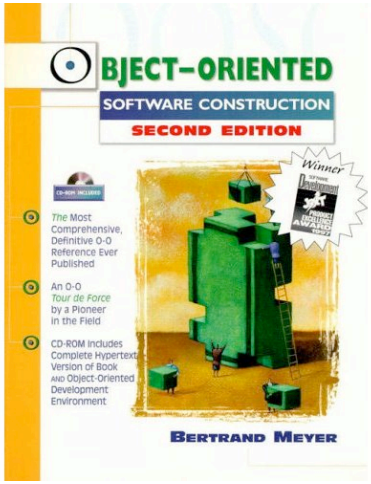


Design Principle / Design Pattern	Abstract Factory	Builder	Factroy Method	Adapter	Bridge	Composite	Decorator	Facade	Proxy	Chain of responsibility	Command	Iterator	Mediator	Observer	State	Strategy	Template Method	Visitor
Dependency Inversion Principle	x		x	x													x	x
Law of Demeter				x			x	x	x				x					
Lose Kopplung	x			x	x	x	x	x	x	x	x	x	x	x				
Hohe Kohäsion															x	x		x
Design by Contract																		
Single Responsibility		x					x		x			x			x	x		
Separation of Concerns		x					x		x	x		x	x		x	x		x
Interface Segregation Principle							x	x										
Liskov Substitutional Principle																		
Open-Closed Principle						x	x			x	x				x	x		

Besten Dank für Ihre Aufmerksamkeit

Quellen

http://de.wikipedia.org/wiki/Prinzipien_Objektorientierten_Designs

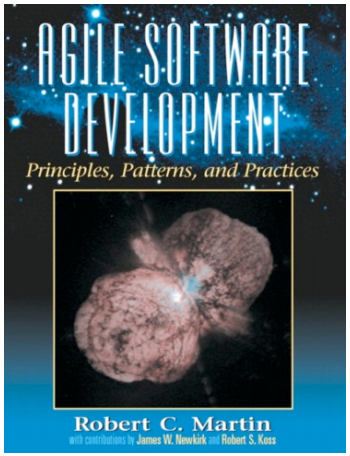


Object-Oriented Software Construction, 1997

ISBN 0-136-29155-4

Agile Software Development Principles, Patterns, and Practice, 2003

ISBN 0135974445



Bildernachweis

Folie Agenda: http://www.kurzreporter.de/wp-content/uploads/2007/12/uhr.jpg
Folie Ziele: http://baltimoreseo-by-murph.com/wp-content/uploads/2009/04/dart_board.jpg
Folie Yoda: http://images.wikia.com/starwars/images/4/45/Yoda.jpg
Folie Wartungskosten: http://users.jyu.fi/~koskinen/smcosts.htm
Folien Dave Nicolette : http://matteo.vaccari.name/tai/slides2010/2010-tai-lezione-01.pdf und
http://www.slideshare.net/xpmatteo/20101125-ocpxpday
Folie zerfallenes Haus: http://www.monolithic.com/stories/uglyhouses/photos/1
Folien DVD Shop: http://www.dvddiamond.com/wp-content/uploads/2010/10/dvd-rental-store.jpg und
http://www.devoted.com.au/rental/images/rentalshop.jpg
Folie Besteck: http://www.dominikundrahel.ch/Bilder/Wunsch/wunsch55.jpg
Folie Modern Times: http://www.follow-me-now.de/assets/images/Moderne_Zeiten-Am_Fliessband.jpg
Folie Dr Evil: http://www.mauchle.name/binarybomb.htm
Folie Mainboard: http://3.bp.blogspot.com/_CfroFM7qQqE/TK0tjKjQfUI/AAAAAAAAACA/Jo6b_oZT3GU/s1600/1071083973.jpg