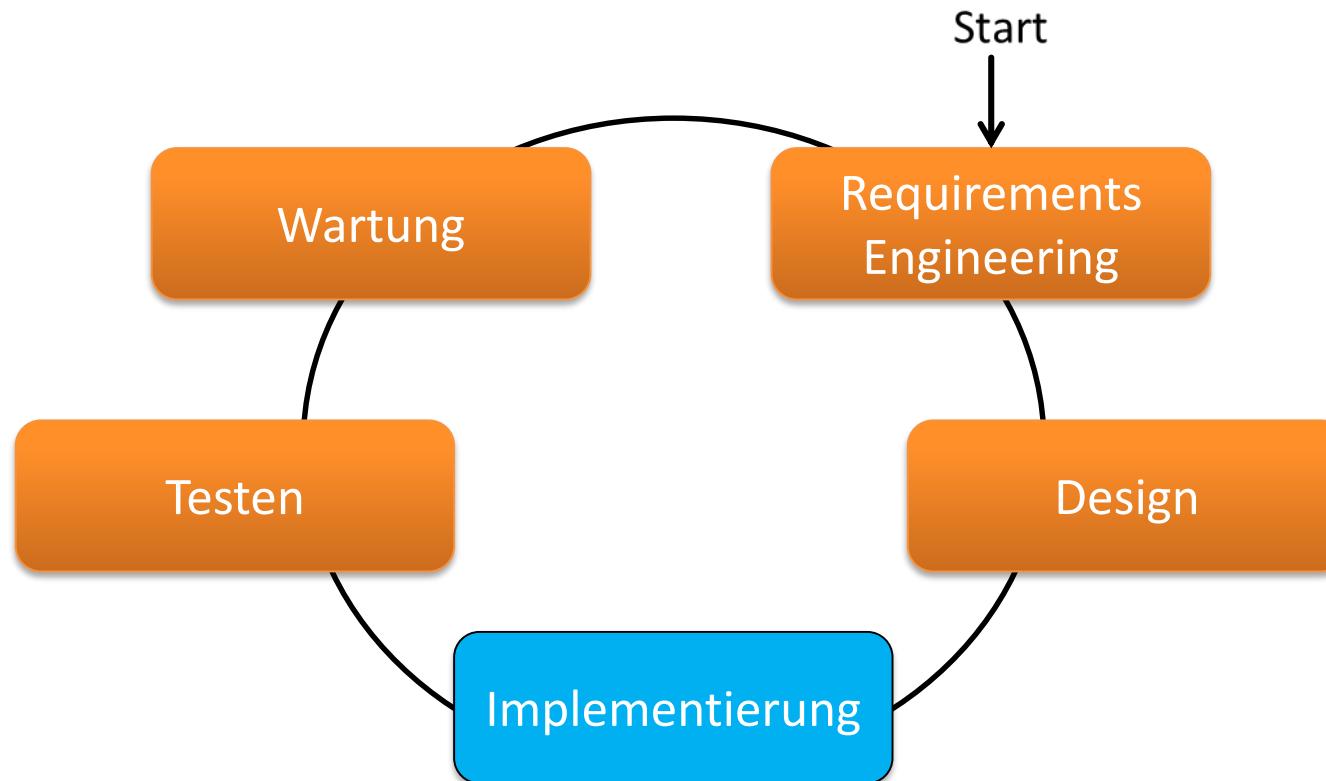


Software Engineering

Implementierung

Authors of slides:
Norbert Siegmund
Janet Siegmund
Oscar Nierstrasz
Sven Apel

Einordnung

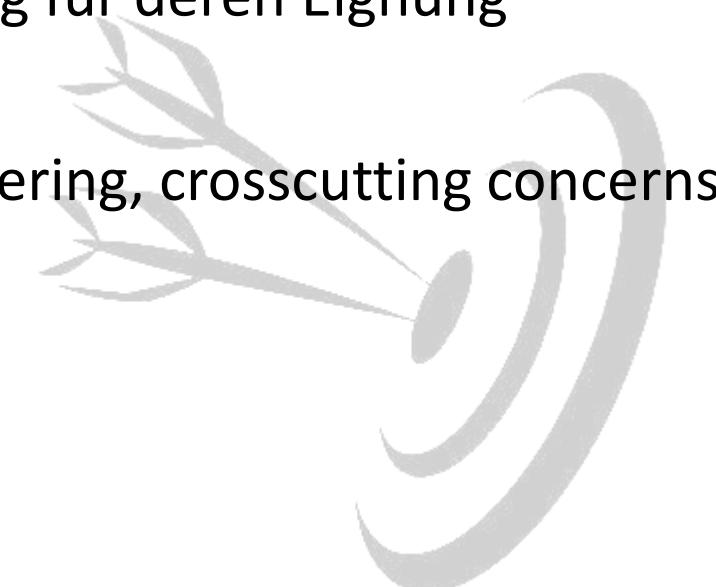


Inhalt

- Probleme bei der Entwicklung konfigurierbarer Software
- State-of-the-Art Ansätze
 - Wie funktionieren sie?
 - Was sind ihre Schwächen?
- Grenzen derzeitiger Techniken

Lernziele

- Erfahren, welche Probleme bei der Entwicklung konfigurierbarer Software existieren
- Kennenlernen von Lösungsansätzen und Abschätzung für deren Eignung
- Was verbirgt sich hinter den Begriffen tangling, scattering, crosscutting concerns, und Tyrannie der dominanten Dekomposition?



Hintergrund Konfigurierbarkeit

Autos - Produktkatalog

BMW Deutschland - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://www.bmw.de/de/de/index_narrowband.html?content=..../de/de/general/configurations_center/configurator.htm

Getting Started Latest Headlines

Home 1 3 5 6 7 X3 X5 Z4 M Gebrauchte Automobile Service & Zubehör Faszination BMW

Mein BMW Kontakt BMW Konfigurator BMW Händler & Service Partner Finanzieren & Versichern Shops Direktabnehmer

BMW Deutschland BMW Freude am Fahren

BMW Konfigurator Zurück

BMW Konfigurator

318i Touring

Gesamtpreis | Monatliche Rate

34.970,00 EUR

Fahrzeugdaten ausdrucken
Finanzierung und Leasing
Beenden und Speichern
Hinweise zu Preisangaben

Modell Grundausstattung Farbe, Interieur + Felgen Editionen + Pakete Sonderausstattungen Zusammenfassung

Getriebe | Klima, Heizung | Komfort/Nutzen | Optik innen/außen | Polsterungen, Sitze | Radio, Audio, Kommunikation, Info | Sicherheit | Sportlichkeit

Ausstattung Code Preis

Getriebe

Automatic Getriebe Bild-[i](#) 205 2.160,00 EUR

Klima, Heizung

Standheizung mit Fernbedienung Bild-[i](#) 536 1.340,00 EUR

Sonnenschutzverglasung, Individual Bild-[i](#) 761 390,00 EUR

Klimaautomatik mit Fondausströmern Bild-[i](#) 534 770,00 EUR Preis-[i](#)

Komfort/Nutzen

Ablagenpaket Bild-[i](#) 493 110,00 EUR

Armauflage vorn, verschiebbar Bild-[i](#) 4AE 150,00 EUR Preis-[i](#)

Autoportiersteuer Bild-[i](#) 313 240,00 EUR

Suche Sitemap Website Einstellungen Weitere BMWWebsites Rechtlicher Hinweis / Impressum Zur Video-Version

Waiting for ecom.bmwgroup.com...

Variantenvielfalt

Beispiel BMW X3



Dachhimmel:
90.000
Varianten-
möglichkeiten

Autotüren:
3.000
Varianten-
möglichkeiten

Hinterachsen:
324
Varianten-
möglichkeiten

„Varianten sind ein wesentlicher Hebel für das Unternehmensergebnis“
— Franz Decker, Leiter Programm Variantenmanagement, BMW Group

PKW-Produktlinien vor 20 Jahren

- Auswahl beschränkte sich auf Autotyp und ggf. noch wenige Extras wie alternativer Kassettenrekorder oder Dachgepäckträger
- Eine Variante (Audi 80, 1.3l, 55PS) machte 40% des Umsatzes aus

PKW-Produktlinien vor wenigen Jahren

- 10^{20} mögliche Varianten eines Audi; 10^{32} mögliche Varianten eines BMW
- Kaum ein Auto verlässt das Werk identisch zu einem vorherigen
- Allein 100 verschiedene Bodengruppen für ein Modell, je nach Motor und Ausstattung
- 50 verschiedene Lenkräder (3 vs. 4 Speichen, Holz vs. Kunststoff vs. Leder, Heizung, Farben)

Warum maßgeschneiderte Software?

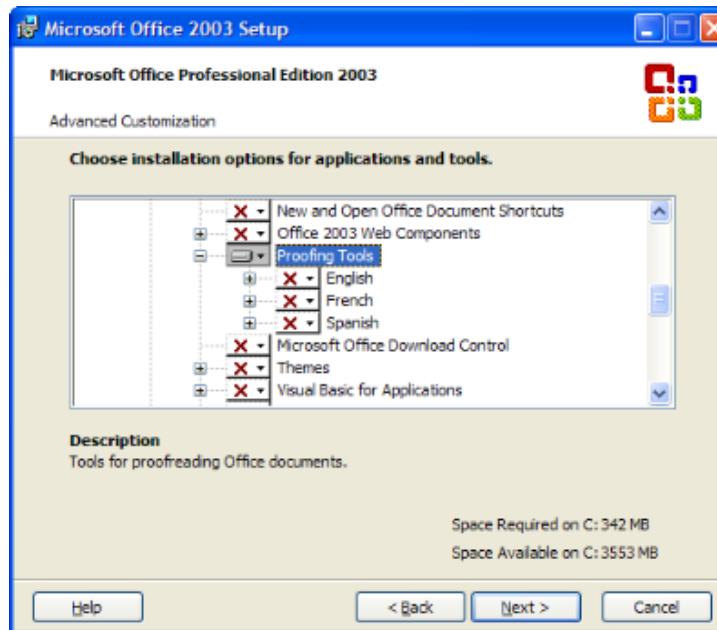
- Ressourcenbeschränkte Systeme
 - Kosten, Energie, Platz,
- Individuelle Systeme versus individuelle Nutzung
 - Ungenutzte Funktionalität als Risiko
 - Wartungs- / Kontroll- / Testaufwand wächst mit Funktionsumfang
- Marketing / Preisdiskriminierung
- Schnellere Reaktion auf Marktveränderungen

Konfigurierbare SW

- Welche konfigurierbaren Softwaresysteme kennen Sie?
 - Nennen Sie Beispiele möglichst unterschiedlicher Domänen
 - Erläutern Sie, warum die Konfigurierbarkeit dort eingeführt wurde

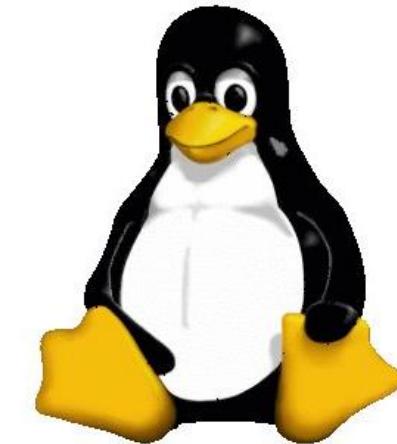


Features in Microsoft Office



Linux-Kernel

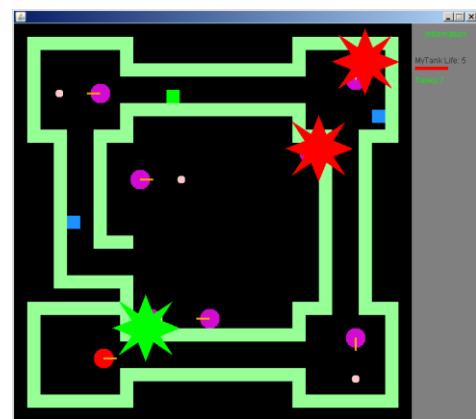
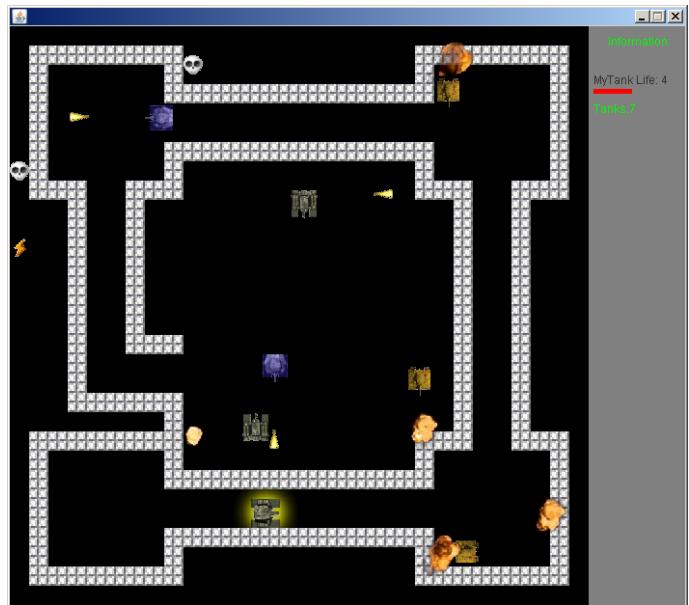
- ca. 6.000.000 Zeilen Quelltext
- Sehr weitgehend konfigurierbar
 - > 10.000 Konfigurationsoptionen! (x86, 64bit, ...)
 - Fast aller Quelltext ist „optional“



Drucker-Firmware



Spiele



Konfigurierbare SW in der Industrie

- HP: Druckertreiber/Firmware
- Nokia: Mobiltelefon-Software, Browser
- Phillips: High-End TVs, Medizintechnik, ...
- TomTom: eingebettete Navigationssysteme
- Cummins: Dieselmotoren-Steuerungssoftware
- LG: Aufzugsteuerungssoftware
- Ericsson: Telecommunication switches
- General Motors: Powertrains
- Viele weitere: Gasturbinen, train control, ship control, frequency converter, internet payment gateway, helicopter avionics software, ...

Herausforderungen



Variabilität = Komplexität

VEGETARIAN

WHICH WICH WOULD YOU LIKE?

- TRIPLE CHEESE MELT
- ELVIS WICH (P.B., Honey & Banana)
- TOMATO & AVOCADO
- BLACK BEAN PATTY
- HUMMUS & BELL PEPPERS

CHOOSE YOUR BREAD

- WHITE
- WHEAT

CHOOSE YOUR CHEESE (Optional)

- AMERICAN
- SWISS
- PROVOLONE
- CHEDDAR
- PEPPER JACK
- MOZZARELLA

How Would You Like Your WICH Worked?

- MUSTARDS
- Yellow
- Dijon
- Honey
- Deli

- MAYOS
- Regular
- Lite
- Horseradish
- Spicy

- SPREADS & SAUCES
- BBQ
- Buffalo
- Marinara
- 1000 Island
- Ranch

- ONIONS
- Red
- Grilled
- Crispy Strings

- VEGGIES
- Lettuce
- Tomato
- Pickles
- Jalapenos
- Olive Salad
- Mushrooms
- Sauerkraut
- Coleslaw
- Bell Peppers

- OILS & SPICES
- Oil
- Vinegar
- Salt
- Pepper
- Oregano
- Parmesan

- EXTRAS (.75¢ Each)
- Bacon
- Avocado
- Pickle (Whole)
- More Meat
- More Cheese

33 Features



eine maßgeschneiderte Variante für
jeden Menschen auf dem Planeten



320 Features
optionale, unabhängige

mehr Varianten als es
Atome im Universum gibt!



2000 Features

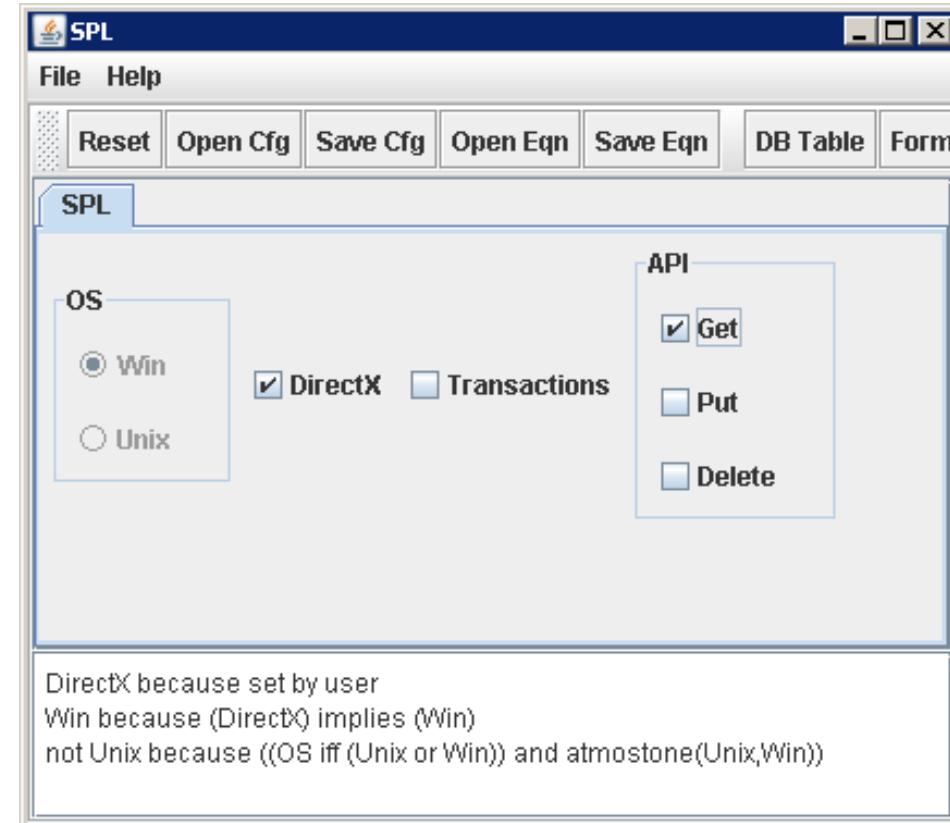
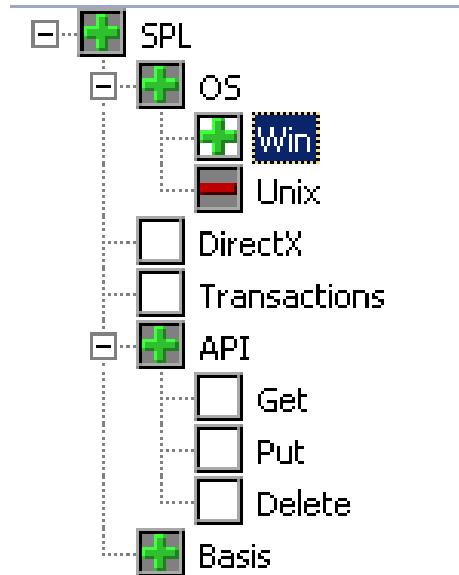
10000 Features



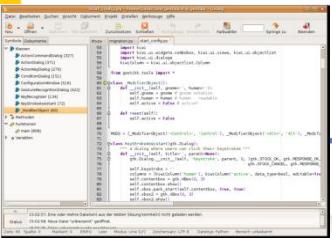
Korrektheit?



Alle Kombinationen sinnvoll?



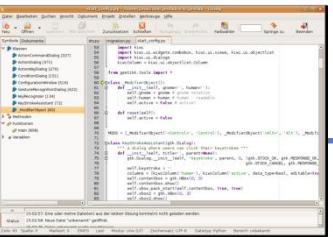
Wiederverwendung bei der Implementierung?



```

public class PrinterDriver {
    public void printDocument(Document document) {
        // Implementation
    }
}

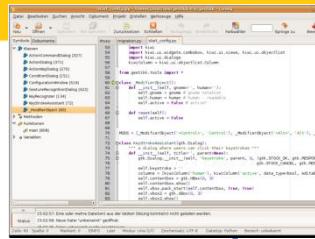
```

```

public class PrinterDriver {
    public void printDocument(Document document) {
        // Implementation
    }
}

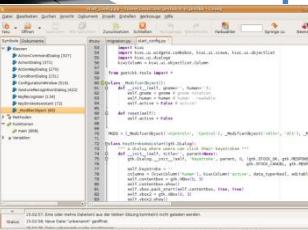
```

```

public class PrinterDriver {
    public void printDocument(Document document) {
        // Implementation
    }
}

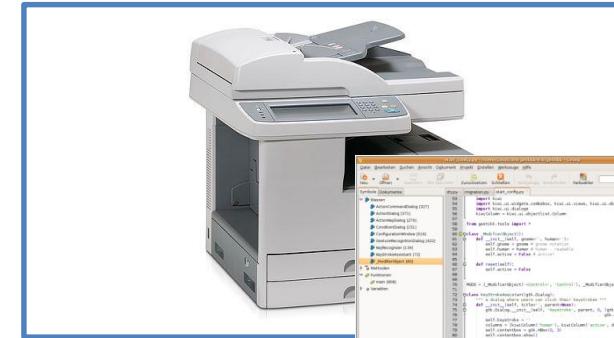
```

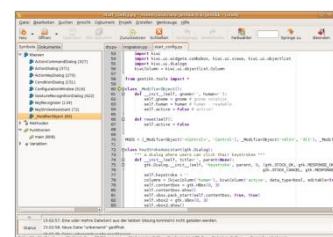
```

public class PrinterDriver {
    public void printDocument(Document document) {
        // Implementation
    }
}

```



Wo Fehler korrigieren?



```

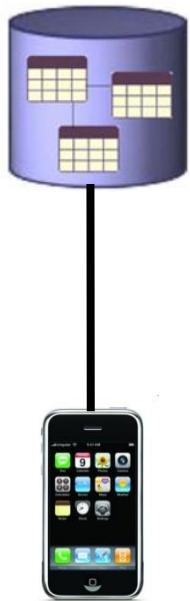
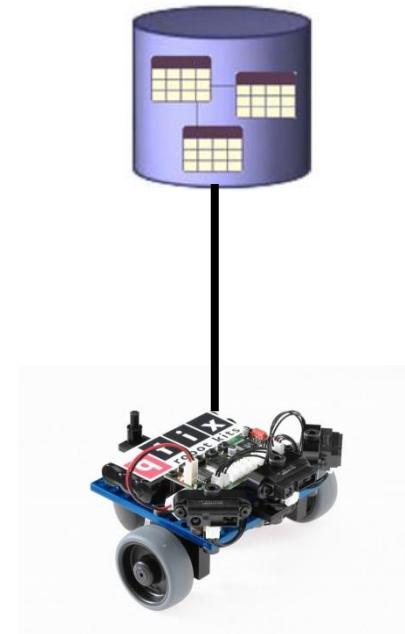
public class PrinterDriver {
    public void printDocument(Document document) {
        // Implementation
    }
}

```



Features...

Beobachtung: Software-Systeme haben Ähnlichkeiten

DBMS₁DBMS₂DBMS₃DBMS₄DBMS₅

Server

Notebook

Smartphone

Sensor

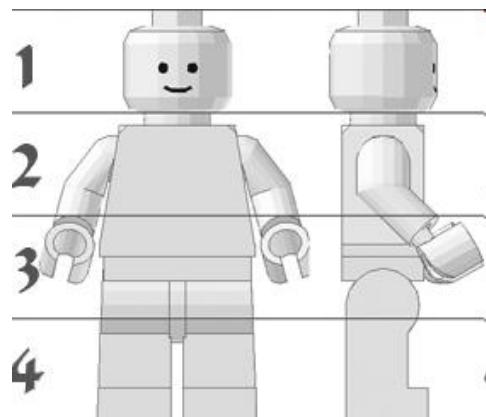
Smartcard

Idee: Ermögliche Wiederverwendung und Variabilität

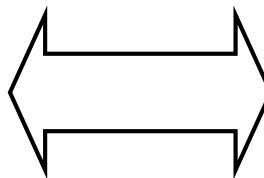
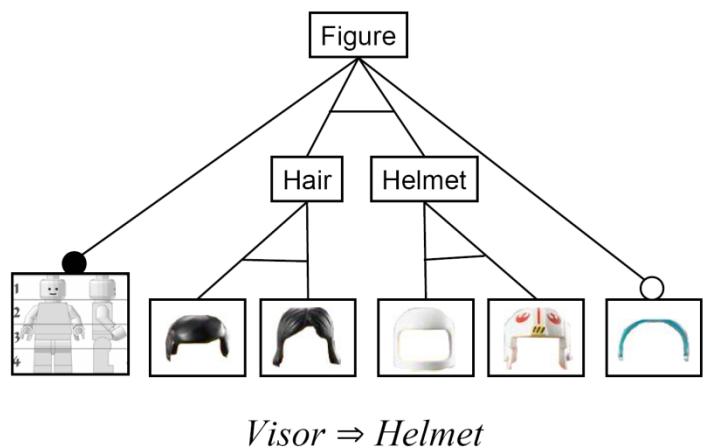


Features

- Was ist ein Feature?
 - Sichtbares Verhalten oder Eigenschaft eines Systems
 - ...dass optional oder alternativ zu anderen Features sein kann
- Features repräsentieren Gemeinsamkeiten und Unterschiede von Software-Systemen



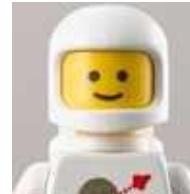
Featureauswahl II



(Figure) ^
(Figure \Leftrightarrow BaseFigure) ^
(Figure \Leftrightarrow Hair v Helmet) ^
(Hair \Leftrightarrow Short v Long) ^
(Helmet \Leftrightarrow Plain v StarWars) ^
(Visor \Rightarrow Figure) ^
(Visor \Rightarrow Helmet)

Produkte und Produktlinien

- Produkt (a.k.a. Variante) \Leftrightarrow valide Featureauswahl



- Produktlinie \Leftrightarrow Menge von gültigen Featureauswahlen in einer Domäne



• • •



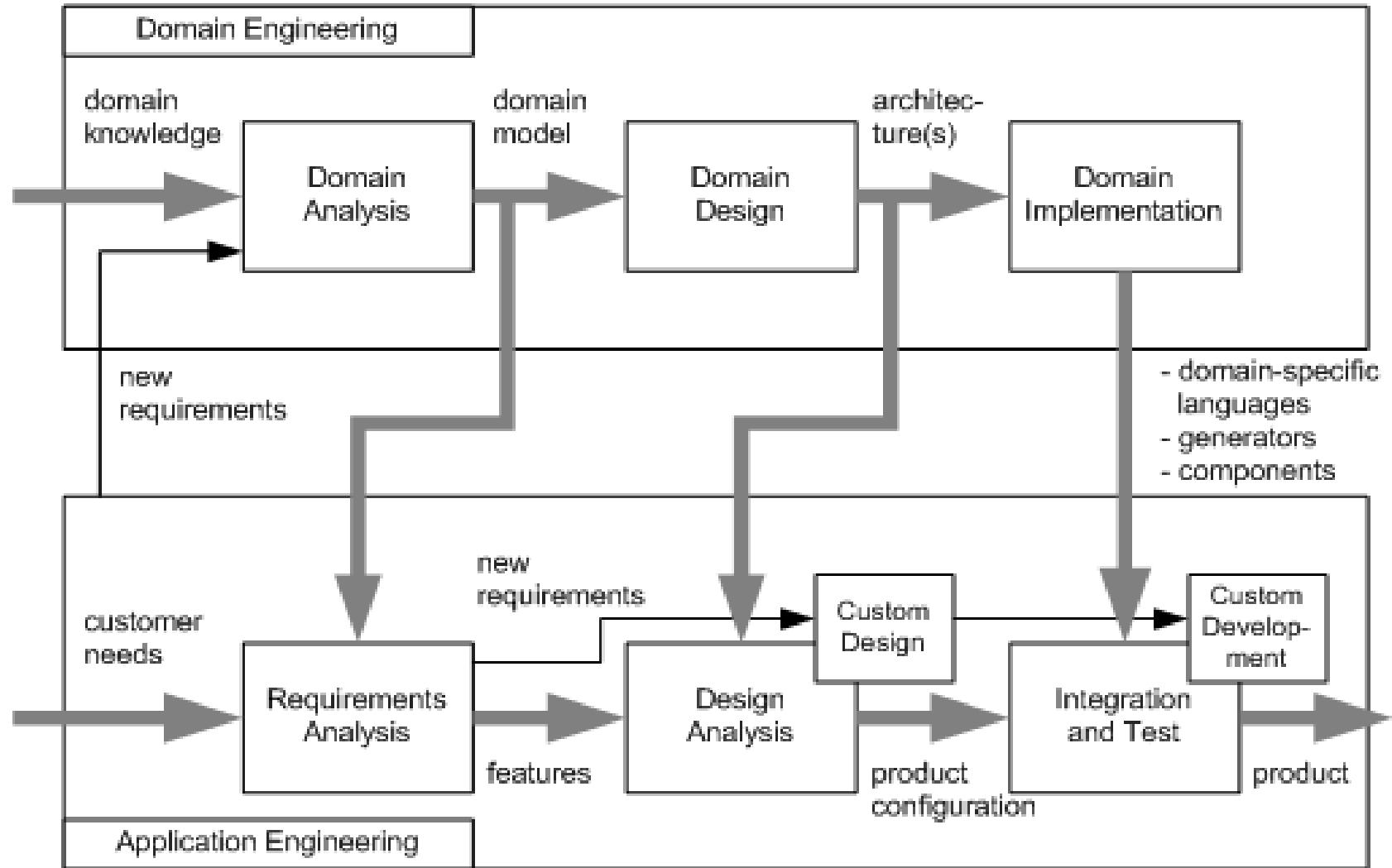
• • •

Domain Engineering

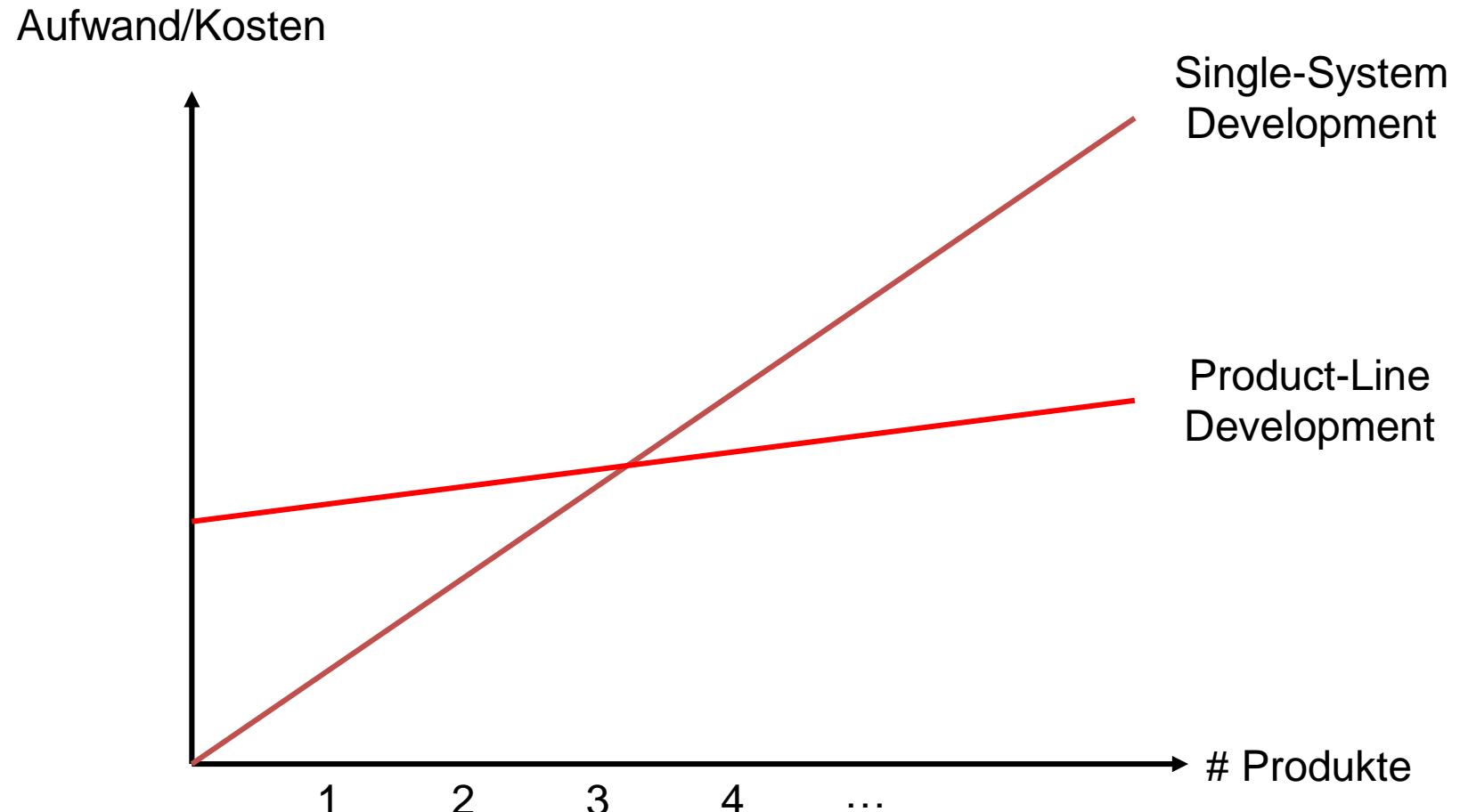
[...] is the activity of collecting, organizing, and storing past experience in building systems [...] in a particular domain in the form of reusable assets [...], as well as providing an adequate means for reusing these assets (i.e., retrieval, qualification, dissemination, adaptation, assembly, and so on) when building new systems.

K. Czarnecki and U. Eisenecker

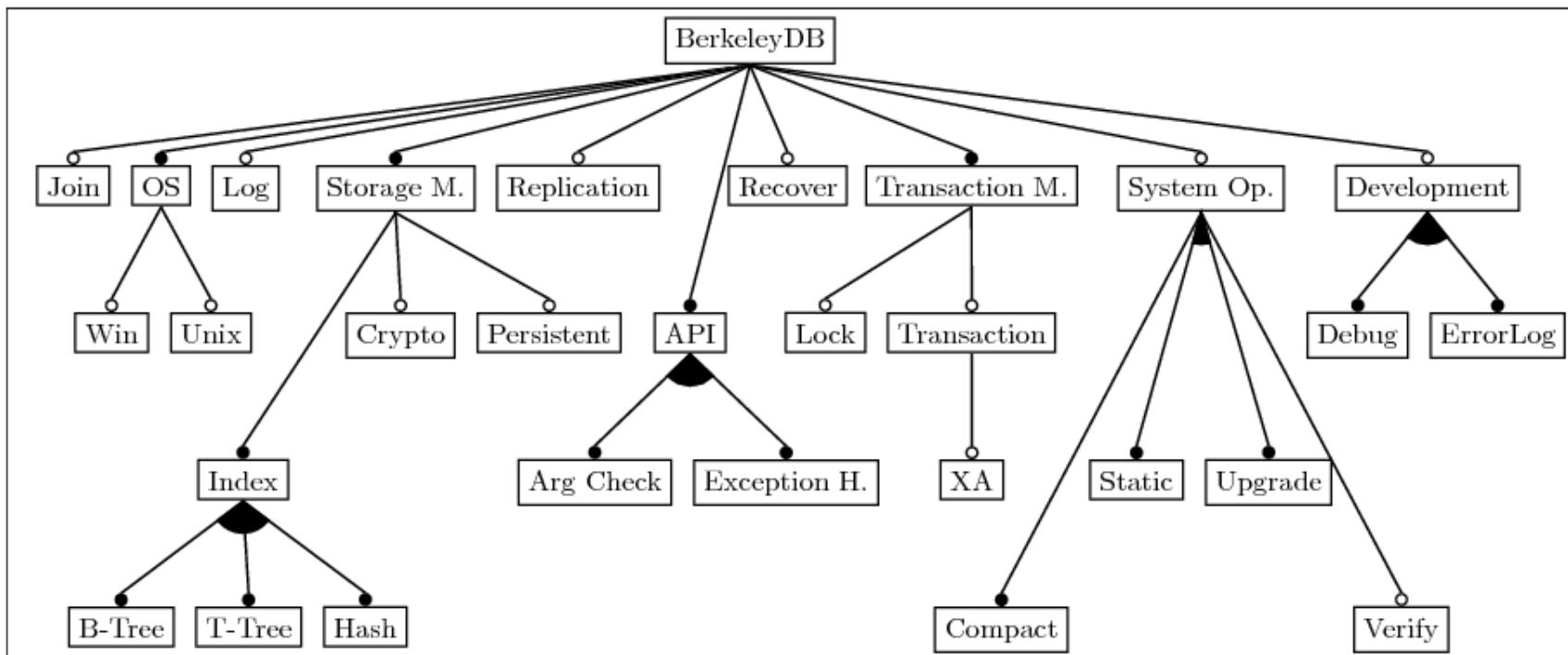
Application and Domain Engineering



Entwicklungsaufwand und -kosten

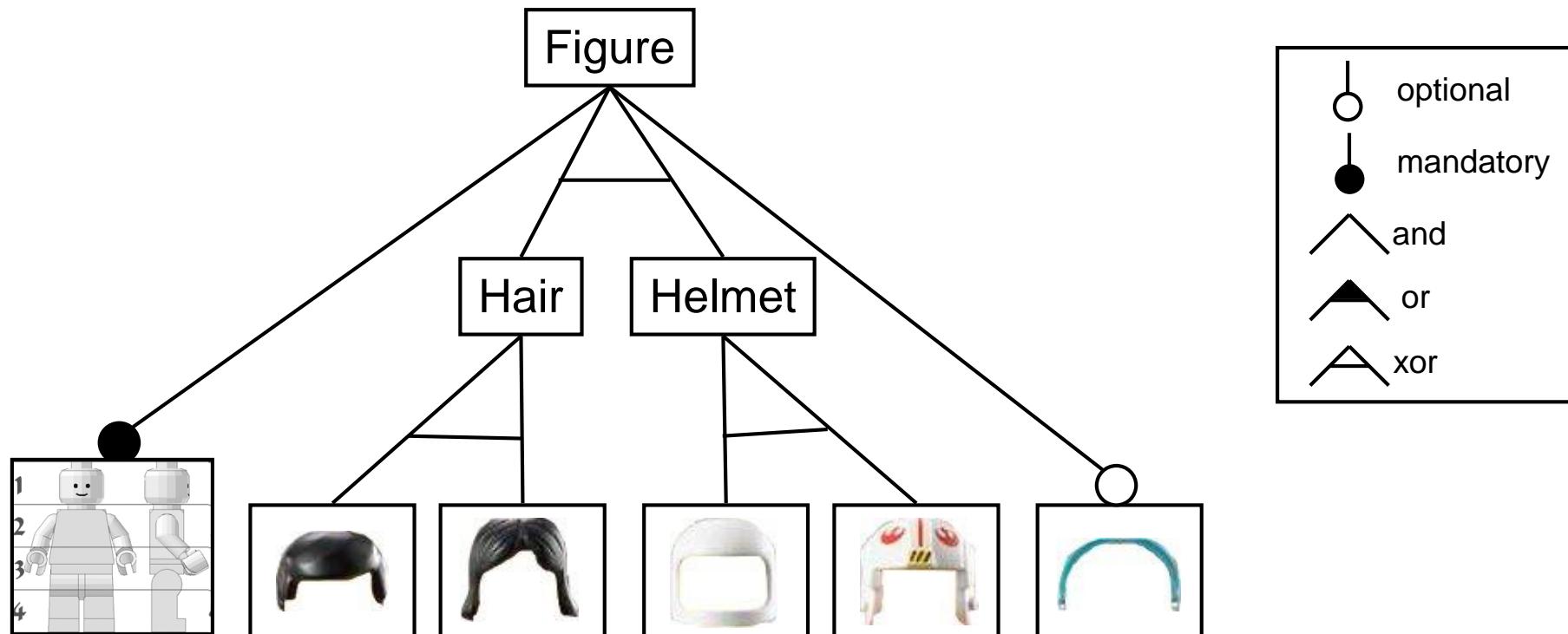


Beispiel: Berkeley DB



Featureauswahl

- Oft ist nur eine Teilmenge aller Kombinationen von Features gültig / sinnvoll

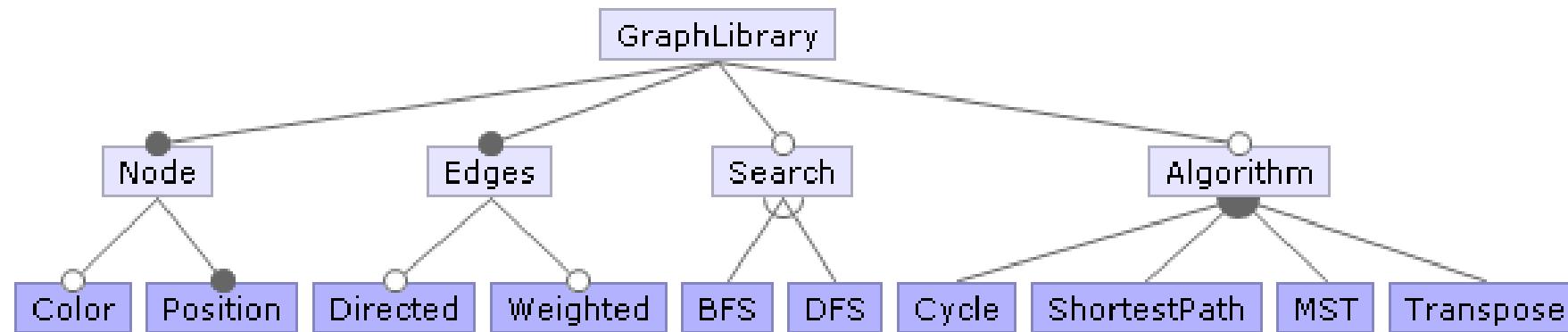


Visor \Rightarrow Helmet

State-of-the-Art: Parameter & Präprozessoren

Beispiel: Graph-Bibliothek

- Bibliothek von Graph-Datenstrukturen und –Algorithmen
 - Gewichtete/ungewichtete Kanten
 - Gerichtete/ungerichtete Kanten
 - Gefärbte Knoten
 - Algorithmen: kürzester Pfad, minimale Spannbäume, Transitive Hülle, ...



Graph-Implementierungsbeispiel

```
class Graph {  
    Vector nv = new Vector(); Vector ev = new Vector();  
    Edge add(Node n, Node m) {  
        Edge e = new Edge(n, m);  
        nv.add(n); nv.add(m); ev.add(e);  
        e.weight = new Weight();  
        return e;  
    }  
    Edge add(Node n, Node m, Weight w)  
    Edge e = new Edge(n, m);  
    nv.add(n); nv.add(m); ev.add(e);  
    e.weight = w; return e;  
}  
void print() {  
    for(int i = 0; i < ev.size(); i++) {  
        ((Edge)ev.get(i)).print();  
    }  
}
```

```
class Node {  
    int id = 0;  
    Color color = new Color();  
    void print() {  
        Color.setDisplayColor(color);  
        System.out.print(id);  
    }  
}
```

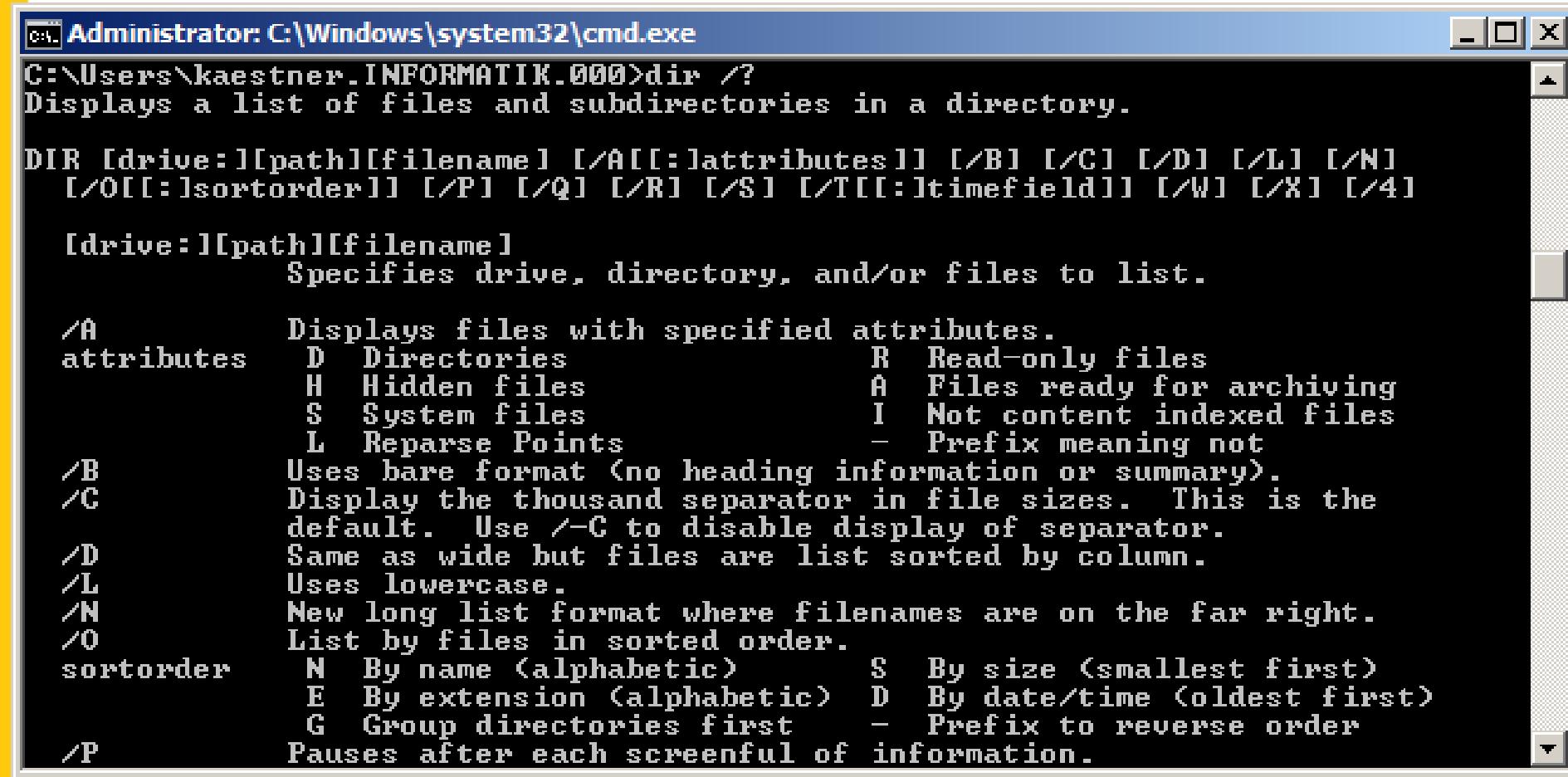
```
class Edge {  
    Node a, b;  
    Color color = new Color();  
    Weight weight = new Weight();  
    Edge(Node _a, Node _b) { a = _a; b = _b; }  
    void print() {  
        Color.setDisplayColor(color);  
        a.print(); b.print();  
        weight.print();  
    }  
}
```

```
class Color {  
    static void setDisplayColor(Color c) { ... }  
}
```

```
class Weight { void print() { ... } }
```

Laufzeit-Parameter

Parameter



Administrator: C:\Windows\system32\cmd.exe

```
C:\>dir /?
Displays a list of files and subdirectories in a directory.

DIR [drive:][path][filename] [/A[[[:l]attributes]] [/B] [/C] [/D] [/L] [/N]
  [/O[[[:l]sortorder]] [/P] [/Q] [/R] [/S] [/T[[[:l]timefield]] [/W] [/X] [/4]

[drive:][path][filename]
  Specifies drive, directory, and/or files to list.

/A      Displays files with specified attributes.
attributes   D  Directories          R  Read-only files
              H  Hidden files        A  Files ready for archiving
              S  System files        I  Not content indexed files
              L  Reparse Points     - Prefix meaning not
/B      Uses bare format (no heading information or summary).
/C      Display the thousand separator in file sizes. This is the
       default. Use /-C to disable display of separator.
/D      Same as wide but files are list sorted by column.
/L      Uses lowercase.
/N      New long list format where filenames are on the far right.
/O      List by files in sorted order.
sortorder   N  By name (alphabetic)    S  By size (smallest first)
            E  By extension (alphabetic) D  By date/time (oldest first)
            G  Group directories first  - Prefix to reverse order
/P      Pauses after each screenful of information.
```

Globale Konfigurationsoptionen

```
class Config {  
    public static boolean isLogging = false;  
    public static boolean isWindows = false;  
    public static boolean isLinux = true;  
}  
class Main {  
    public void foo() {  
        if (isLogging)  
            log("running foo()");  
        if (isWindows)  
            callWindowsMethod();  
        else if (isLinux)  
            callLinuxMethod();  
        else  
            throw RuntimeException();  
    }  
}
```

```
class Conf {
    public static boolean COLORED = true;
    public static boolean WEIGHTED = false;
}
```

Graph-Implementierung

```
class Graph {
    Vector nv = new Vector(); Vector ev = new Vector();
    Edge add(Node n, Node m) {
        Edge e = new Edge(n, m);
        nv.add(n); nv.add(m); ev.add(e);
        if (███████████) e.weight = new Weight();
        return e;
    }
    Edge add(Node n, Node m, Weight w) {
        if (███████████) throw RuntimeException();
        Edge e = new Edge(n, m);
        nv.add(n); nv.add(m); ev.add(e);
        e.weight = w; return e;
    }
    void print() {
        for(int i = 0; i < ev.size(); i++) {
            ((Edge)ev.get(i)).print();
        }
    }
}
```

```
class Node {
    int id = 0;
    Color color = new Color();
    void print() {
        if (███████████) Color.setDisplayColor(color);
        System.out.print(id);
    }
}
```

```
class Edge {
    Node a, b;
    Color color = new Color();
    Weight weight;
    Edge(Node _a, Node _b) { a = _a; b = _b; }
    void print() {
        if (Conf.COLORED) Color.setDisplayColor(color);
        a.print();
        if (Conf.WEIGHTED) weight.print();
    }
}
```

```
class Color {
    static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

Propagierte Parameter

```
* the doc templates in the doc_src directory.  
*/  
public Sequence openSequence(Transaction txn,  
                           DatabaseEntry key,  
                           SequenceConfig config)  
throws DatabaseException {  
  
    checkEnv();  
    DatabaseUtil.checkForNullDbt(key, "key", true);  
    checkRequiredDbState(OPEN, "Can't call Database.openSequence");  
    checkWritable("openSequence");  
    trace(Level.FINEST, "Database.openSequence", txn, key, null, null);  
  
    return new Sequence(this, txn, key, config);  
}  
  
/**  
 * Javadoc for this public method is generated via  
 * the doc templates in the doc_src directory.  
 */  
public void removeSequence(Transaction txn, DatabaseEntry key)  
throws DatabaseException {  
  
    delete(txn, key);  
}  
  
public synchronized Cursor openCursor(Transaction txn,  
                                     CursorConfig cursorConfig)  
throws DatabaseException {
```

durch viele Aufrufe
propagiert statt
globaler Variable

```
* the doc templates in the doc_src directory.  
*/  
public Sequence openSequence(Transaction txn,  
                           DatabaseEntry key,  
                           SequenceConfig config)  
throws DatabaseException {  
  
    checkEnv();  
    DatabaseUtil.checkForNullDbt(key, "key", true);  
    checkRequiredDbState(OPEN, "Can't call Database.openSequence");  
    checkWritable("openSequence");  
    trace(Level.FINEST, "Database.openSequence", txn, key, null, null);  
  
    return new Sequence(this, txn, key, config);  
}  
  
/**  
 * Javadoc for this public method is generated via  
 * the doc templates in the doc_src directory.  
 */  
public void removeSequence(Transaction txn, DatabaseEntry key)  
throws DatabaseException {  
  
    delete(txn, key);  
}
```

Diskussion

- Variabilität im gesamten Program verteilt
- Globale Variablen oder lange Parameterlisten
- Konfiguration geprüft?
- Änderungen zur Laufzeit möglich?
- Geschützt vor Aufruf deaktivierter Funktionalität?
- Kein Generator; immer alle Variabilität ausgeliefert
 - Codegröße, Speicherverbrauch, Performance
 - Ungenutzte Funktionalität als Risiko

Präprozessoren

- Transformieren Quelltext vor Compileraufruf
- Von einfachen #include Befehlen und bedingter Übersetzung zu komplexen Makrosprachen und Regeln
- In vielen Programmiersprachen üblich
 - C, C++, Fortran, Erlang mit eigenem Präprozessor
 - C#, Visual Basic, D, PL/SQL, Adobe Flex

#ifdef Beispiel aus Berkeley DB

```
static int __rep_queue_filedone(dbenv, rep, rfp)
    DB_ENV *dbenv;
    REP *rep;
    __rep_fileinfo_args *rfp; {
#ifndef HAVE_QUEUE
    COMPQUIET(rep, NULL);
    COMPQUIET(rfp, NULL);
    return (__db_no_queue_am(dbenv));
#else
    db_pgno_t first, last;
    u_int32_t flags;
    int empty, ret, t_ret;
#endif
#ifndef DIAGNOSTIC
    DB_MSGBUF mb;
#endif
    // over 100 lines of additional code
}
```

Präprozessor in Java?

- Nicht nativ vorhanden
- Bedingte Kompilierung in manchen Compilern nur auf Statement-Ebene, nicht für Klassen oder Methoden

```
class Example {  
    public static final boolean DEBUG = false;  
  
    void main() {  
        System.out.println("immer");  
        if (DEBUG)  
            System.out.println("debug info");  
    }  
}
```

- Externe Tools vorhanden, z.B. CPP, Antenna, Munge, XVCL, Gears, pure::variants

Munge

- Simpler Präprozessor für Java Code
- Ursprünglich für Swing in Java 1.2

```
class Example {  
void main() {  
    System.out.println("immer");  
    /*if[DEBUG]*/  
    System.out.println("debug info");  
    /*end[DEBUG]*/  
}  
}
```

java Munge **-DDEBUG -DFEATURE2** Datei1.java Datei2.java ... Zielverzeichnis

Beispiel: Konfigurierbarer Graph

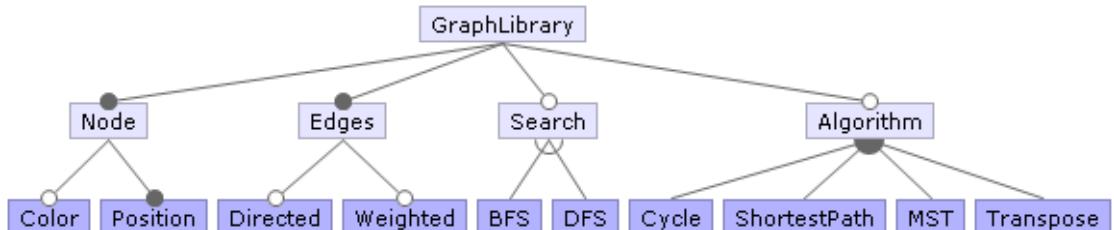
```
class Graph {
    Vector nv = new Vector(); Vector ev = new Vector();
    Edge add(Node n, Node m) {
        Edge e = new Edge(n, m);
        nv.add(n); nv.add(m); ev.add(e);
        e.weight = new Weight();
        return e;
    }
    Edge add(Node n, Node m, Weight w)
        Edge e = new Edge(n, m);
        nv.add(n); nv.add(m); ev.add(e);
        e.weight = w; return e;
    }
    void print() {
        for(int i = 0; i < ev.size(); i++) {
            ((Edge)ev.get(i)).print();
        }
    }
}
```

```
class Node {
    int id = 0;
    Color color = new Color();
    void print() {
        Color.setDisplayColor(color);
        System.out.print(id);
    }
}
```

```
class Edge {
    Node a, b;
    Color color = new Color();
    Weight weight = new Weight();
    Edge(Node _a, Node _b) { a = _a; b = _b; }
    void print() {
        Color.setDisplayColor(color);
        a.print(); b.print();
        weight.print();
    }
}
```

```
class Color {
    static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```



Graph-Beispiel mit Munge

```
class Graph {  
    Vector nv = new Vector(); Vector ev = new Vector();  
    Edge add(Node n, Node m) {  
        Edge e = new Edge(n, m);  
        nv.add(n); nv.add(m); ev.add(e);  
        /*if[WEIGHT]*/  
        e.weight = new Weight();  
        /*end[WEIGHT]*/  
        return e;  
    }  
    /*if[WEIGHT]*/  
    Edge add(Node n, Node m, Weight w)  
    Edge e = new Edge(n, m);  
    nv.add(n); nv.add(m); ev.add(e);  
    e.weight = w; return e;  
}  
/*end[WEIGHT]*/  
void print() {  
    for(int i = 0; i < ev.size(); i++) {  
        ((Edge)ev.get(i)).print();  
    }  
}
```

```
class Edge {  
    Node a, b;  
    /*if[COLOR]*/  
    Color color = new Color();  
    /*end[COLOR]*/  
    /*if[WEIGHT]*/  
    Weight weight;  
    /*end[WEIGHT]*/  
    Edge(Node _a, Node _b) { a = _a; b = _b; }  
    void print() {  
        /*if[COLOR]*/  
        Color.setDisplayColor(color);  
        /*end[COLOR]*/  
        a.print(); b.print();  
        /*if[WEIGHT]*/  
        weight.print();  
        /*end[WEIGHT]*/  
    }  
}
```

```
/*if[WEIGHT]*/  
class Weight { void print() { ... } }  
/*end[WEIGHT]*/
```

```
/*if[COLOR]*/  
class Color {  
    static void setDisplayColor(Color c) { ... } }  
/*end[COLOR]*/
```

```
class Node {  
    int id = 0;  
    /*if[COLOR]*/  
    Color color = new Color();  
    /*end[COLOR]*/  
    void print() {  
        /*if[COLOR]*/  
        Color.setDisplayColor(color);  
        /*end[COLOR]*/  
        System.out.print(id);  
    }  
}
```

Diskussion

Vorteile von Präprozessoren

- In vielen Sprachen bereits enthalten / simple Tools
- Den meisten Entwicklern bereits bekannt
- Sehr einfaches Programmierkonzept:
markieren und entfernen
- Sehr flexibel / ausdrucksstark
- Nachträgliche Einführung von Variabilität in bestehendes Projekt einfach

Problem: Unlesbarer Quelltext

- Vermischung von zwei Sprachen (C und #ifdef, oder Java und Munge, ...)
- Kontrollfluss schwer nachvollziehbar
- Lange Annotationen schwer zu finden
- Zusätzliche Zeilenumbrüche zerstören Layout

-> Features lieber trennen?

```
class Stack {  
    void push(Object o  
    #ifdef SYNC  
    , Transaction txn  
    #endif  
    ) {  
        if (o==null  
    #ifdef SYNC  
        || txn==null  
    #endif  
        ) return;  
    #ifdef SYNC  
        Lock l=txn.lock(o);  
    #endif  
        elementData[size++] = o;  
    #ifdef SYNC  
        l.unlock();  
    #endif  
        fireStackChanged();  
    } }  
}
```

Präprozessor in Femto OS

```
femtoos_app.c femtoos_core.c femtoos_port.c femtoos_shared.c
void privTaskInit(Tuint8 uiTaskNumber)
{
    privTrace(traceTaskInit | uiTaskNumber);
    TtaskControlBlock * taskTCB = privGetTaskControlBlock(uiTaskNumber);
    #if (defReUseTaskInit == cfgTrue)
        if ((uiInitControl & defInitLockMask) != 0)
        {
            #if (cfgUseSynchronization != cfgFalse)
                if (uiTaskNumber < defNumberOfTasks)
                {
                    privCleanSlotStack((TtaskExtension *)taskTCB);
                    #if ((defUseMutexes == cfgTrue) & (defUseFileSystem == cfgFalse))
                        privReleaseSyncBlockingTask();
                    #endif
                }
            #endif
        }
    #endif
}
```

Probleme von Präprozessoren

- Komplexität durch beliebige Schachtelung
- Fehleranfälligkeit durch Komplexität und unkontrollierten undisziplinierten Einsatz
- Beispiele:
 - Variabler Rückgabetyp

```
Edge/*if[WEIGHT]*/Weight/*end[WEIGHT]*/ add(Node n, Node m /*if[WEIGHT]*/, int  
w/*end[WEIGHT]*/ {  
    return new Edge/*if[WEIGHT]*/Weight/*end[WEIGHT]*/(n, m /*if[WEIGHT]*/,  
w/*end[WEIGHT]*/);  
}
```

- Komma bei mehreren Parametern

```
Edge set(/*if[WEIGHT]*/int w/*if(COLOR)*/, /*end(COLOR)/*end[WEIGHT]*/  
/*if(COLOR)*/int c/*end(COLOR)*/ {  
    ...  
}
```

Problem: Fehleranfällig

- Syntaxfehler

```
static int _rep_queue_filedone(...)  
    DB_ENV *dbenv;  
    REP *rep;  
    __rep_fileinfo_args *rfp; {  
#ifndef HAVE_QUEUE  
    COMPQUIET(rep, NULL);  
    COMPQUIET(rfp, NULL);  
    return (__db_no_queue_am(dbenv));  
#else  
    db_pgno_t first, last;  
    u_int32_t flags;  
    int empty, ret, t_ret;  
#ifdef DIAGNOSTIC  
    DB_MSGBUF mb;  
#endif  
//over 100 lines of additional code  
}  
#endif
```

- Typfehler

```
#ifdef TABLES  
class Table {  
    void insert(Object data,  
               Txn txn) {  
        storage.set(data,  
                    txn.getLock());  
    }  
}  
#endif  
class Storage {  
#ifdef WRITE  
    boolean set(...) { ... }  
#endif  
}
```

Probleme von Präprozessoren II

- Feature-Code ist komplett verstreut
 - -> Feature-Traceability-Problem
 - Wie findet man einen Fehler im *Weight* Feature?
- Verhindert/erschwert Tool Support
 - Erfahrungen aus C/C++ (Refactoring, Analyse, ...)
 - Munge und andere: Definition in Kommentaren

Eine Frage der Größe

ApplicationSession



StandardSession



ServerSession



SessionInterceptor



StandardManager



StandardSessionManager



ServerSessionManager



Limitierungen:

Querschneidende Belange (Crosscutting Concerns)

Querschneidende Belange

- Engl. crosscutting concerns
- Behauptung: Nicht alle Belange (Features) in einem Programm können mittels Objekten (Komponenten, Klassen) modularisiert werden
- Belange sind semantisch zusammenhängende Einheiten
- Aber ihre Implementierung ist manchmal verstreut, vermischt und repliziert im Code

Beispiel Querschneidende Belange

- Code für verschiedene Belange vermischt
- Code repliziert
- Im Beispiel Operationen modular, aber Sperren, Logging, Puffer und Authentifizierung nicht

```
class DatabaseApplication
//... Datenfelder
//... Logging Stream
//... Cache Status
public void authorizeOrder(
    Data data, User currentUser, ...){
    // prüfe Autorisierung
    // Objekt sperren für Synchronisation
    // Aktualität des Puffers prüfen
    // Start der Operation loggen
    // eigentliche Operation ausführen
    // Ende der Operation loggen
    // Sperre auf Objekt freigeben
}
public void startShipping(
    OtherData data, User currentUser, ...){
    // prüfe Autorisierung
    // Objekt sperren für Synchronisation
    // Aktualität des Puffers prüfen
    // Start der Operation loggen
    // eigentliche Operation ausführen
    // Ende der Operation loggen
    // Sperre auf Objekt freigeben
}
```

Verstreuter Code

Code Scattering

The diagram illustrates the concept of code scattering by showing five separate code snippets in boxes:

- Graph Class:**

```
class Graph {  
    Vector nv = new Vector(); Vector ev = new Vector();  
    Edge add(Node n, Node m) {  
        Edge e = new Edge(n, m);  
        nv.add(n); nv.add(m); ev.add(e);  
        if (Conf.WEIGHTED) e.weight = new Weight();  
        return e;  
    }  
  
    Edge add(Node n, Node m, Weight w)  
    if (!Conf.WEIGHTED) throw RuntimeException();  
    Edge e = new Edge(n, m);  
    nv.add(n); nv.add(m); ev.add(e);  
    e.weight = w; return e;  
}  
  
void print() {  
    for(int i = 0; i < ev.size(); i++) {  
        ((Edge)ev.get(i)).print();  
    }  
}
```
- Node Class:**

```
class Node {  
    String id = "0";  
    Color color = new Color();  
  
    if (Conf.COLORED) Color.setDisplayColor(color);  
    System.out.print(id);  
}
```
- Edge Class:**

```
class Edge {  
    Node a, b;  
    Color color = new Color();  
    Weight weight;  
    Edge(Node _a, Node _b) { a = _a; b = _b; }  
  
    void print() {  
        if (Conf.COLORED) Color.setDisplayColor(color);  
        a.print(); b.print();  
        if (!Conf.WEIGHTED) weight.print();  
    }  
}
```
- Color Class:**

```
class Color {  
    static void setDisplayColor(Color c) { ... }  
}
```
- Weight Class:**

```
class Weight { void print() { ... } }
```

Red arrows point from the `if (Conf.WEIGHTED)` check in the `Graph` class to the `Weight` class, and from the `if (Conf.COLORED)` check in the `Node` class to the `Color` class, illustrating how code fragments are scattered across different classes.

Vermischter Code

```
class Graph {
    Vector nv = new Vector(); Vector ev = new Vector();
    Edge add(Node n, Node m) {
        Edge e = new Edge(n, m);
        nv.add(n); nv.add(m); ev.add(e);
        if (Conf.WEIGHTED) e.weight = new Weight();
        return e;
    }
    Edge add(Node n, Node m, Weight w)
        if (!Conf.WEIGHTED) throw RuntimeException();
        Edge e = new Edge(n, m);
        nv.add(n); nv.add(m); ev.add(e);
        e.weight = w; return e;
    }
    void print() {
        for (Edge e : edges)
            System.out.print(e);
    }
}
```

Code Tangling

```
class Node {
    int id = 0;
    Color color = new Color();
    void print() {
        if (Conf.COLORED) Color.setDisplayColor(color);
        System.out.print(id);
    }
}
```

```
class Edge {
    Node a, b;
    Color color = new Color();
    Weight weight;
    Edge(Node _a, Node _b) { a = _a; b = _b; }
    void print() {
        if (Conf.COLORED) Color.setDisplayColor(color);
        a.print(); b.print();
        if (!Conf.WEIGHTED) weight.print();
    }
}
```

```
class Color {
    static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

Scattering und Tangling

- Verstreuter Code (code scattering)
 - Code, der zu einem Belang gehört, ist nicht modularisiert, sondern im gesamten Programm verteilt
 - Häufig kopierter Code (auch wenn es je nur ein einzelner Methodenaufruf ist)
 - Oder stark verteilte Implementierung von (komplementären) Teilen eines Belangs
- Vermischter Code (code tangling)
 - Code, der zu verschiedenen Belangen gehört, ist in einem Modul (oder einer Methode) vermischt

Probleme querschneidender Belange

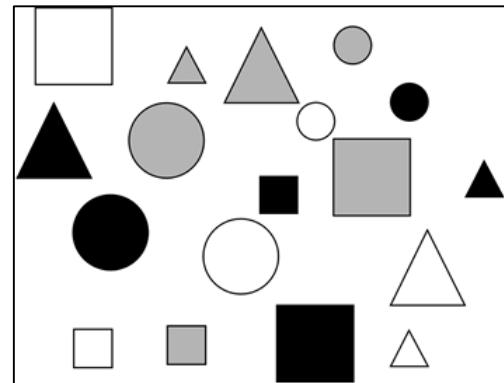
- Belange verschwinden in der Implementierung
 - Was gehört alles zu einem Belang?
 - Bei Wartungsaufgaben muss der ganze Quelltext durchsucht werden
- Schwierige Arbeitsteilung
 - Für unterschiedliche Belange kann es unterschiedliche Experten geben; alle müssen am gleichen Code-Fragment arbeiten
- Geringere Produktivität, schwierige Evolution
 - Beim Hinzufügen neuer Funktionalitäten muss sich der Entwickler um diverse andere Belange kümmern, die erstmal nur ablenken (Lesbarkeit, Erfassbarkeit)

Tyrannie der dominanten Dekomposition

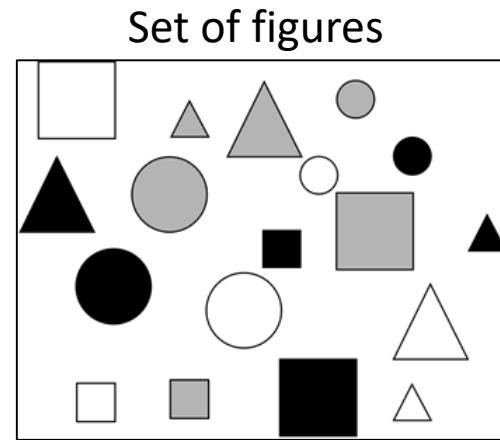
- Viele Belange können modularisiert werden, jedoch nicht immer gleichzeitig
 - Problemstellung nur in einer Richtung modularisierbar
 - Im Graph können Farben modularisiert werden...
 - ...dann sind die Datenstrukturen (Node, Edge) verteilt
- Entwickler wählen eine Dekomposition aus (z.B. Operationen, Authentifizierung, Datenstrukturen), aber einige andere Belange „schneiden quer“
- **Gleichzeitige Modularisierung entlang verschiedener Dimensionen nicht möglich**

Tyrannie der dominanten Dekomposition

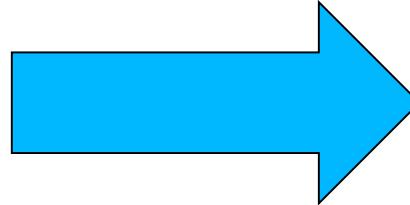
Set of figures



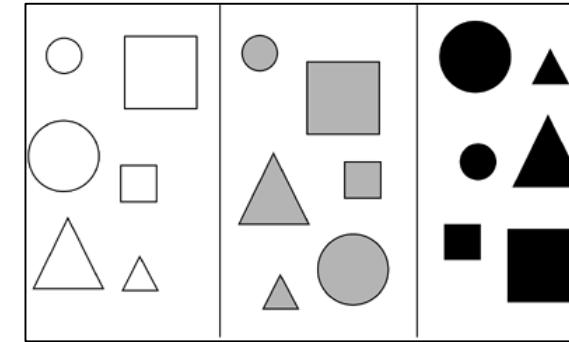
Tyrannie der dominanten Dekomposition



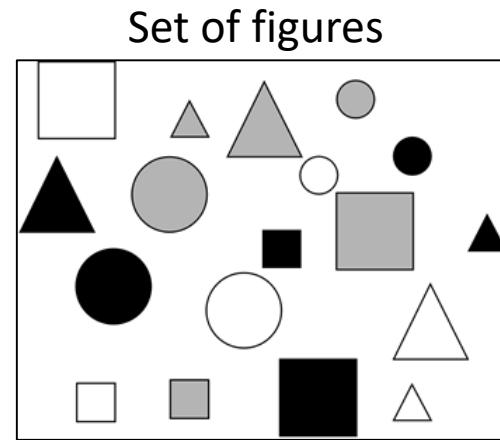
Decomposition



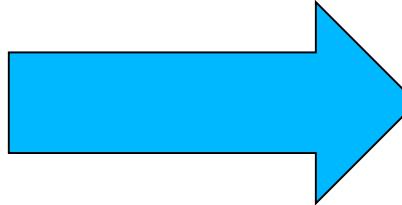
By color



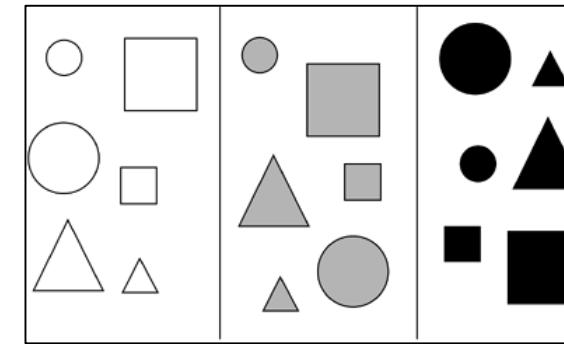
Tyrannie der dominanten Dekomposition



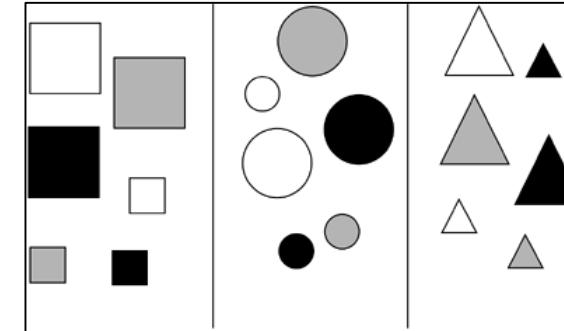
Decomposition



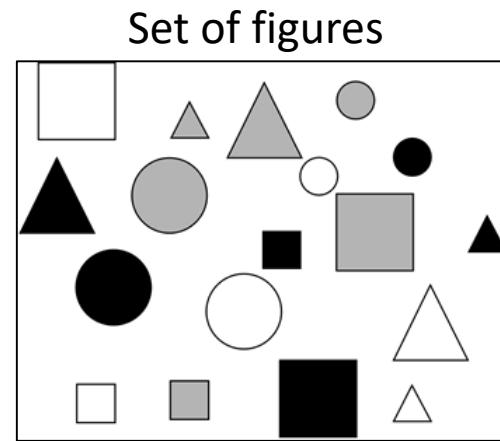
By color



By shape



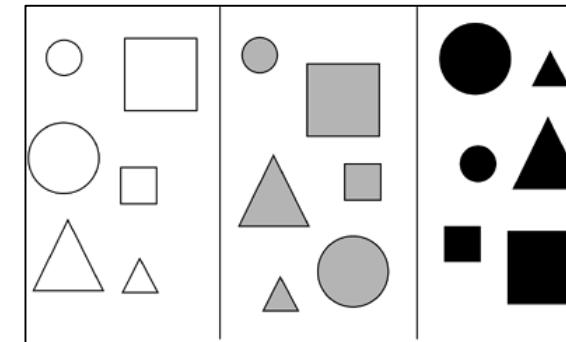
Tyrannie der dominanten Dekomposition



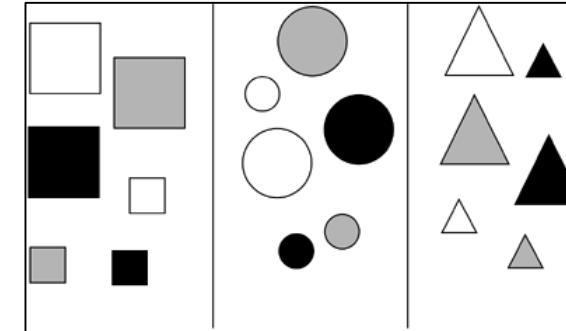
Decomposition



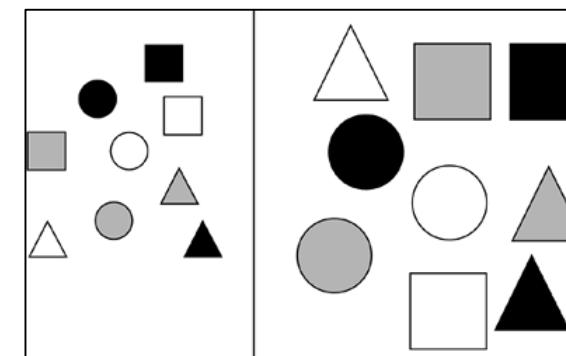
By color



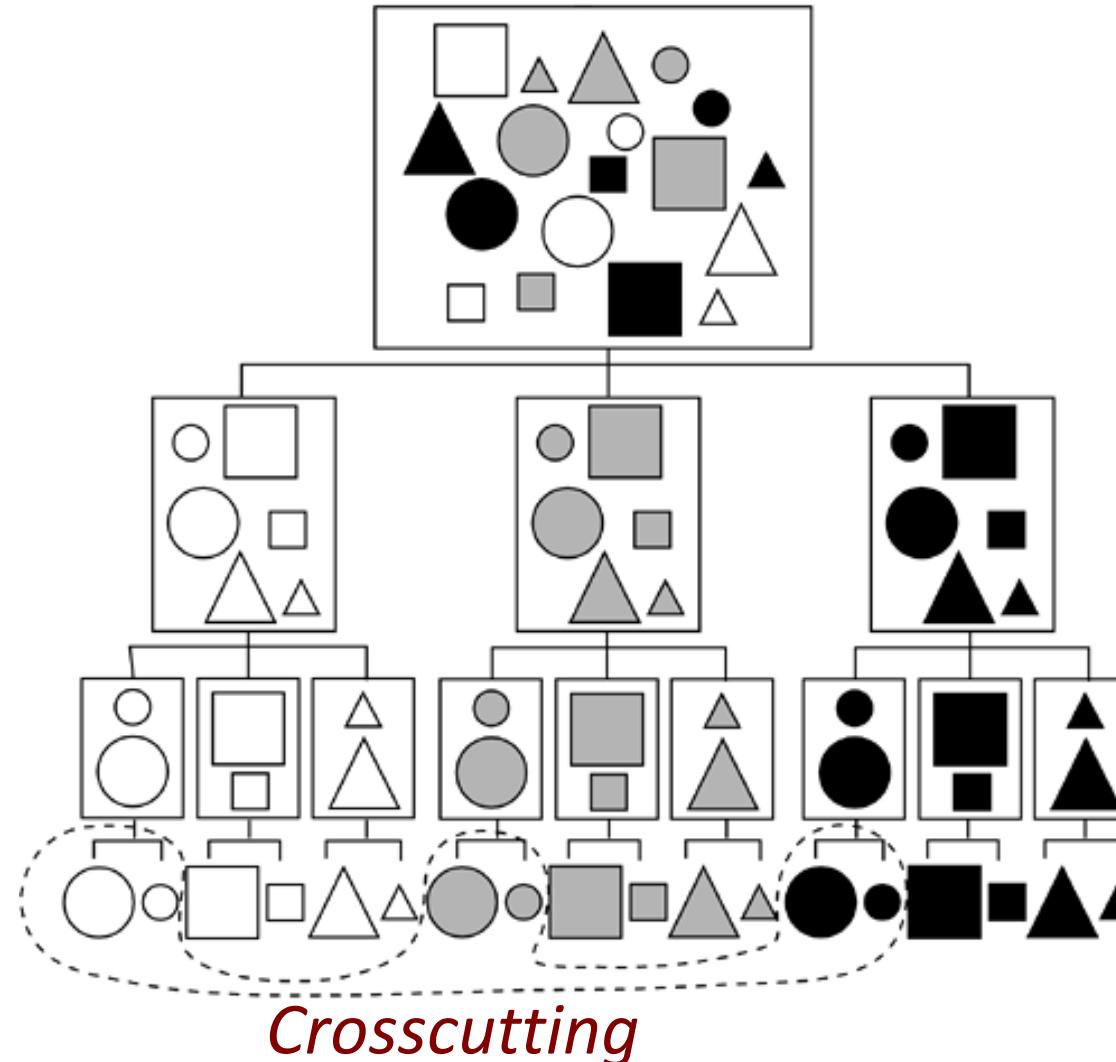
By shape



By size



Tyrannie der dominanten Dekomposition

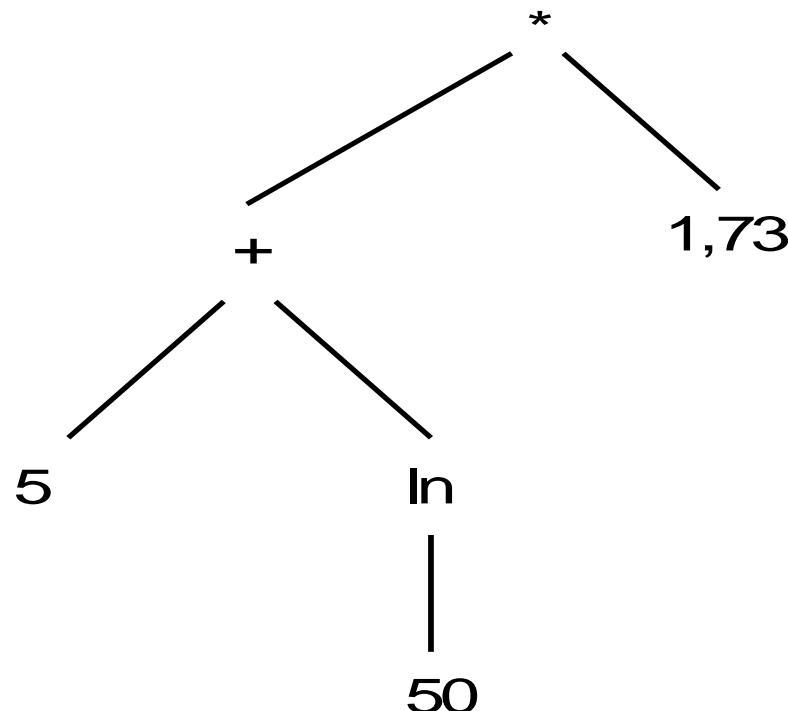


Beispiel: Expression Problem

- Frage: Wie weit kann man Datenstrukturen und Methoden abstrahieren, so dass man beide unabhängig erweitern kann...
 - ohne bestehenden Code zu ändern (oder sogar ohne neu-compilieren des bestehenden Codes)
 - mehrfach, in beliebiger Reihenfolge und
 - ohne (nicht-triviale) Code-Replikation

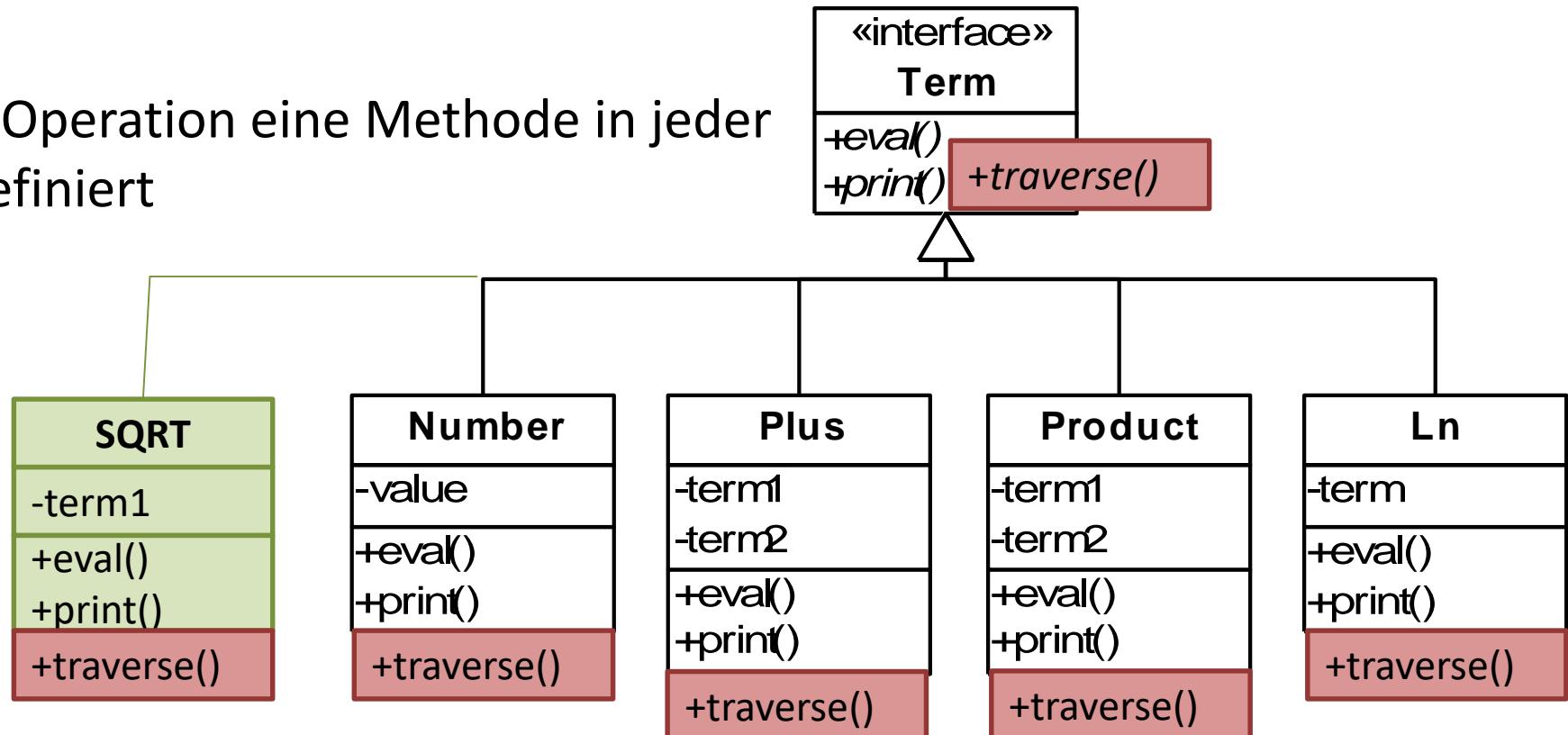
Ausdrücke (expressions)

- Aufgabe: Mathematische Ausdrücke werden in einer Baumstruktur gespeichert und können ausgerechnet und ausgegeben werden



Implementierung 1: Daten-zentriert

- Rekursive Klassenstruktur (Composite Pattern)
- Für jede Operation eine Methode in jeder Klasse definiert

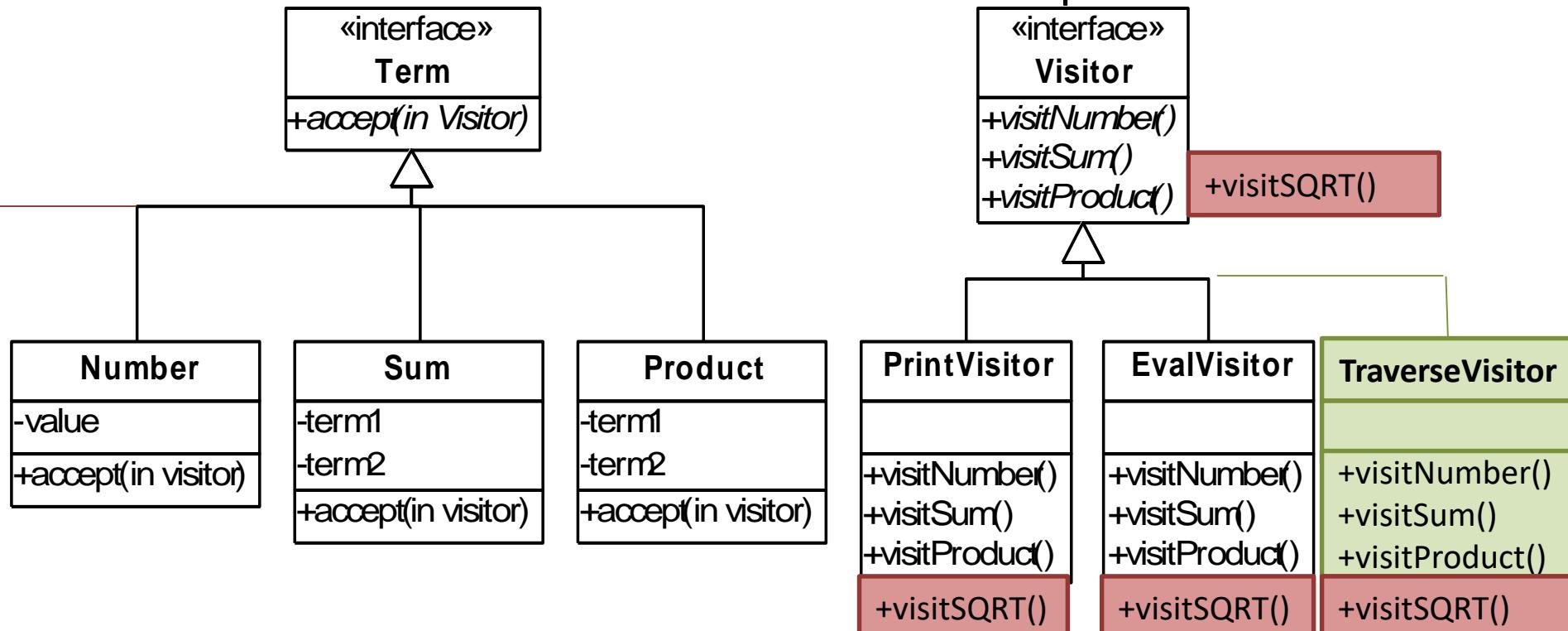


Problem von Implementierung 1

- Ausdrücke sind modular
- Neue Operationen, z. B. `drawTree` oder `simplify`, können nicht einfach hinzugefügt werden
- Alle bestehenden Klassen müssen angepasst werden!
- Operationen sind querschneidend zu den Ausdrücken

Implementierung 2: Methoden-zentriert

- Nur eine Methode accept-Methode pro Klasse
- Methoden werden mit dem Visitor-Pattern implementiert



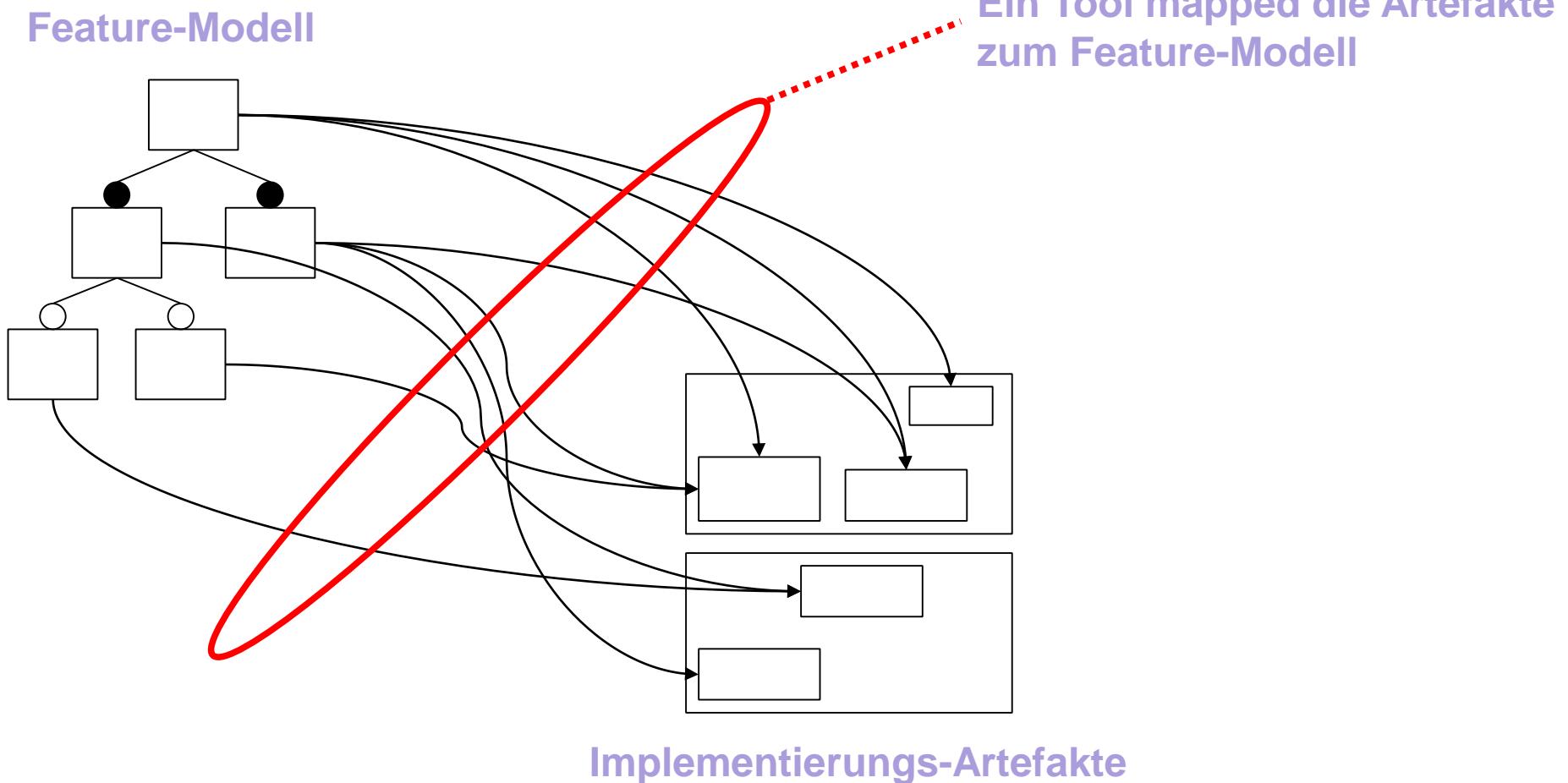
Problem von Implementierung 2

- Operationen sind modular
- Neue Ausdrücke, z. B. Min oder Power können nicht einfach hinzugefügt werden
- Für jede neue Klasse müssen alle Visitor-Klassen angepasst werden
- Ausdrücke sind querschneidend zu den Operationen

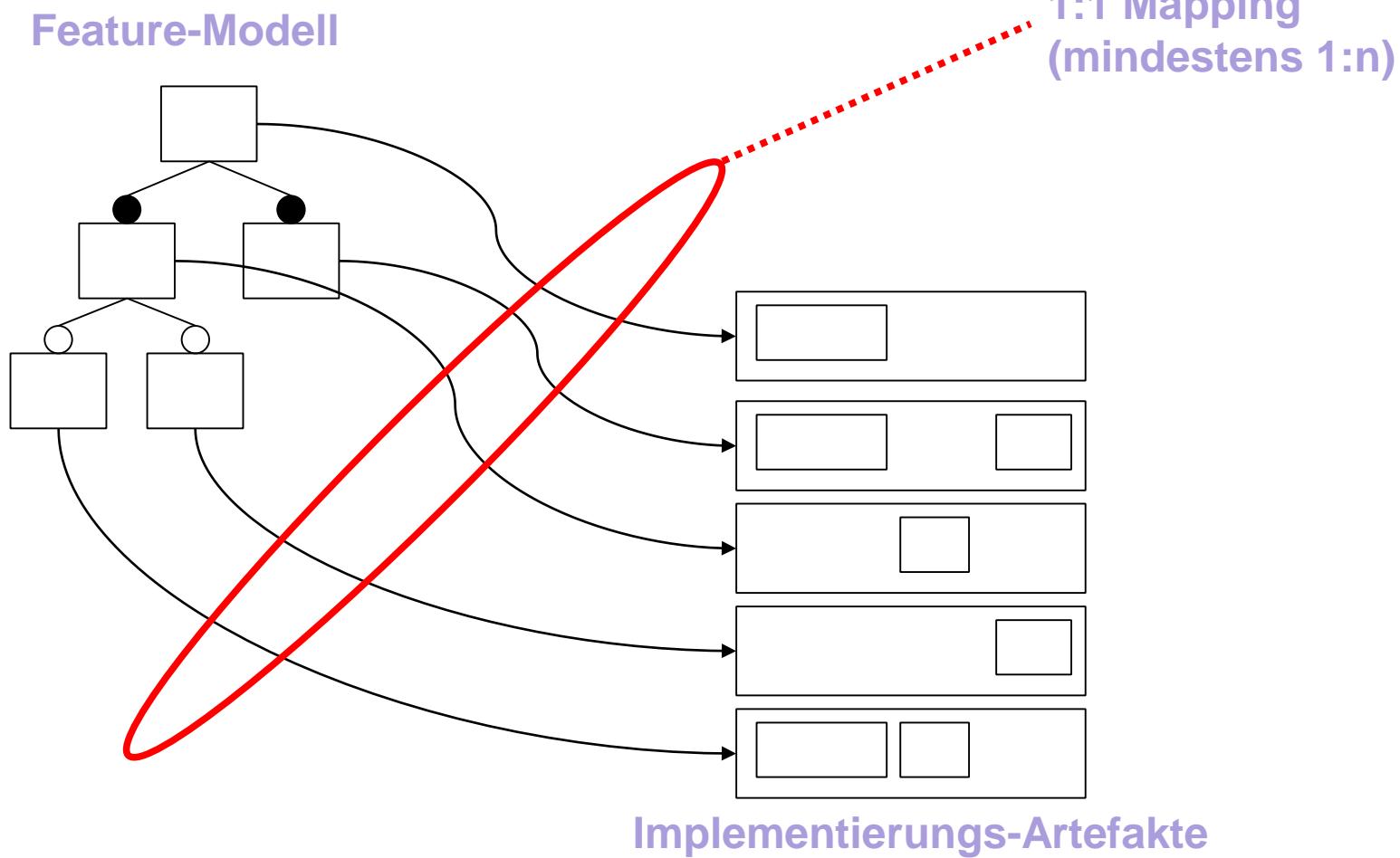
Diskussion: Expression Problem

- Nur sehr schwer möglich Ausdrücke und Operationen darauf gleichzeitig zu modularisieren
- Daten-zentrierter Ansatz
 - Neue Ausdrücke können direkt hinzugefügt werden: modular
 - Neue Operationen müssen in alle Klassen eingefügt werden: nicht modular
- Methoden-zentrierter Ansatz
 - Neue Operationen können als weiterer Visitor hinzugefügt werden: modular
 - Für neue Klassen müssen alle bestehenden Visitors erweitert werden: nicht modular

Feature Traceability with Tool Support



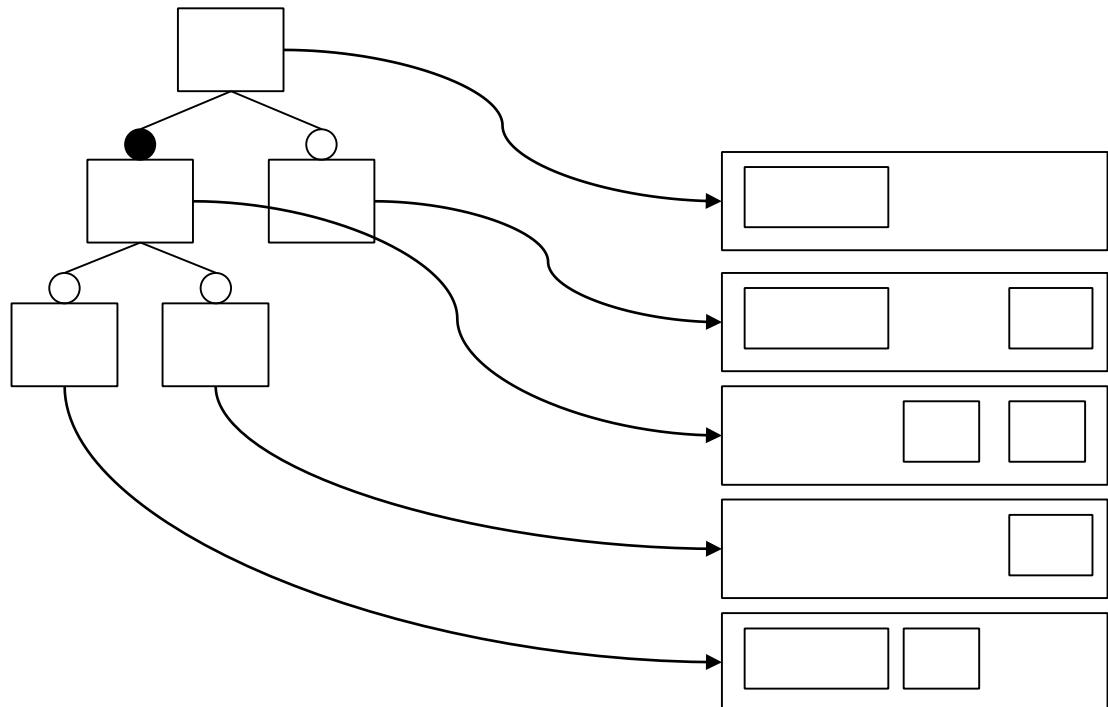
Feature Traceability durch Sprach-Unterstützung



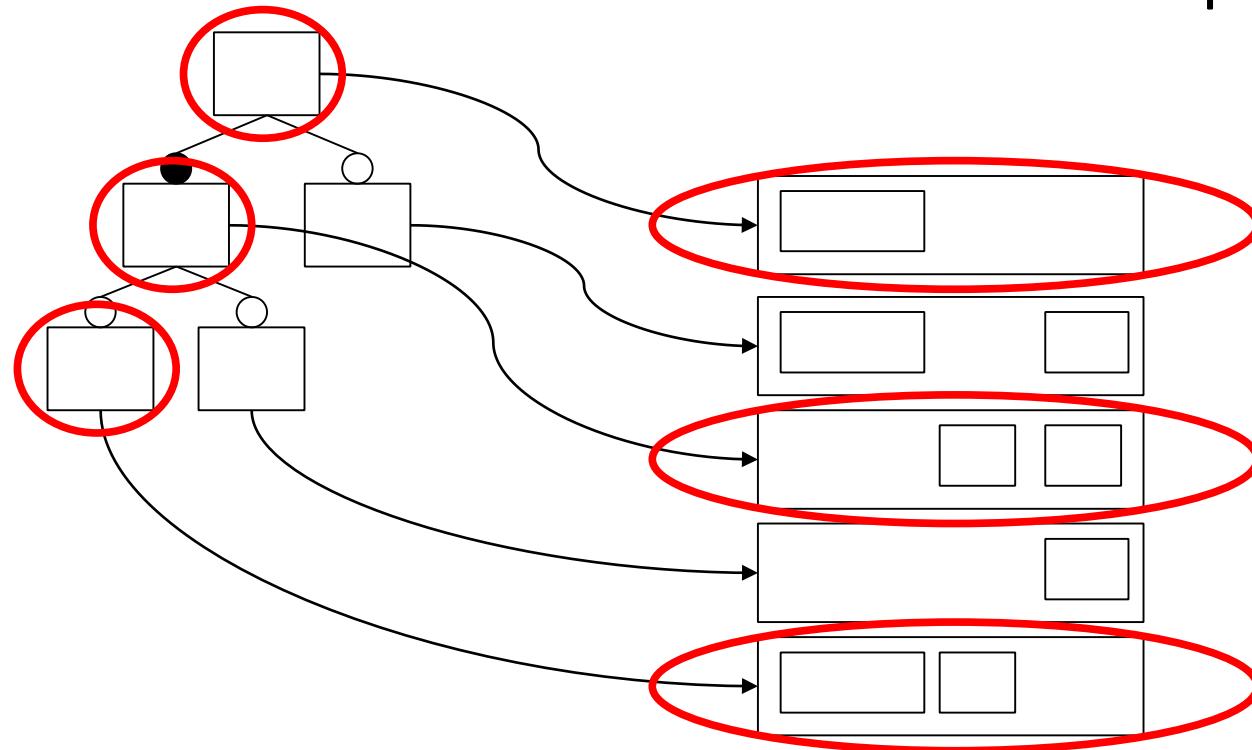
Feature-Oriented Programming

- Prehofer, ECOOP'97 und Batory, ICSE'03
- Sprach-basierter Ansatz um das Feature-Traceability-Problem zu lösen
- Jedes Feature ist in einem Feature-Modul implementiert
 - Dadurch kann jedes Feature gut überblickt werden
 - Features sind getrennt in Modulen
 - Features können einfach zusammengesetzt werden
- Feature-basierte Programm-Generierung

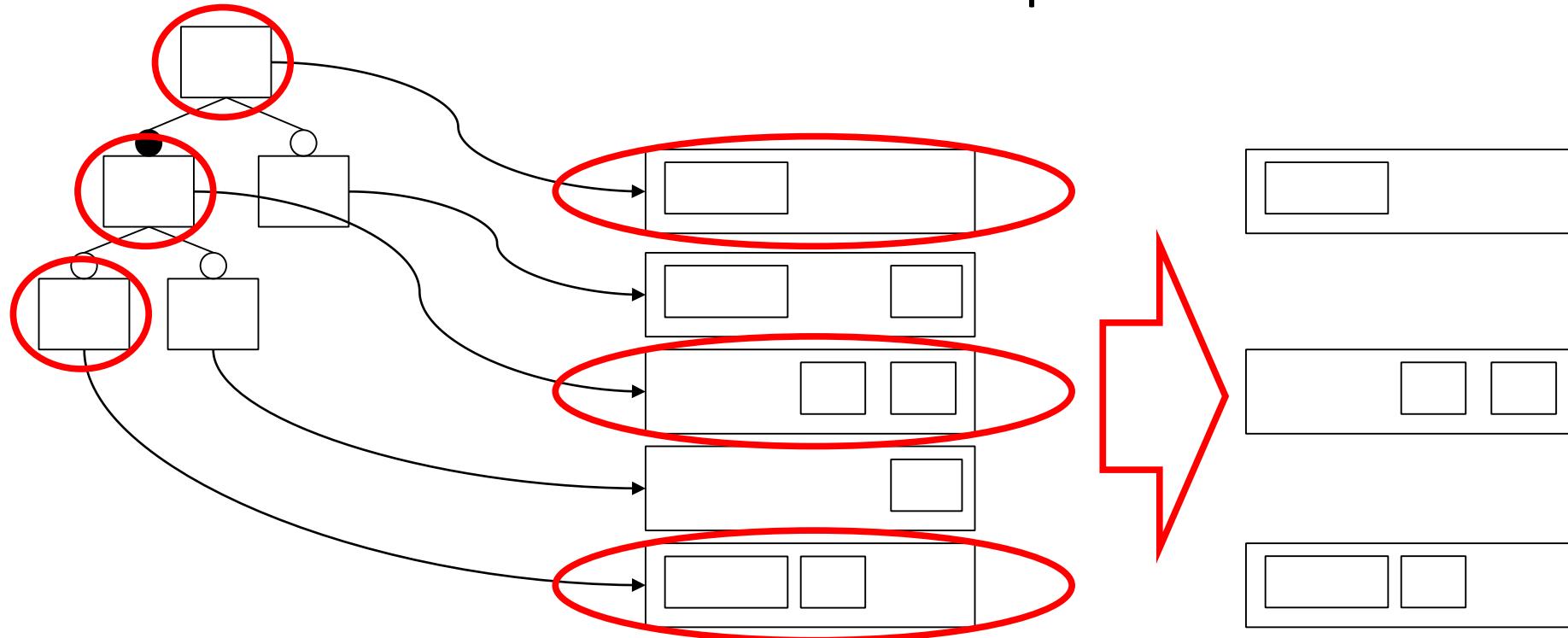
Feature-Komposition



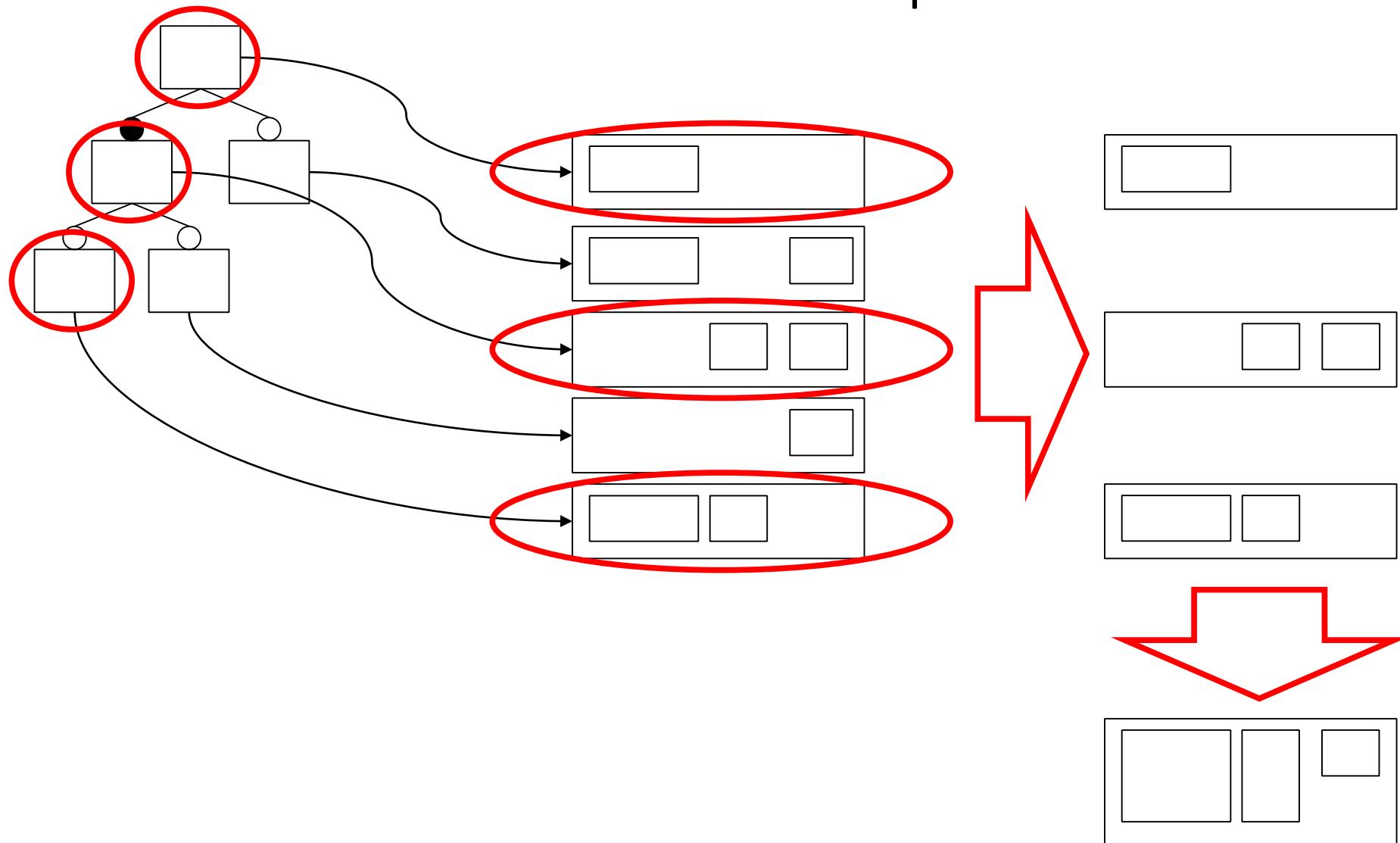
Feature-Komposition



Feature-Komposition

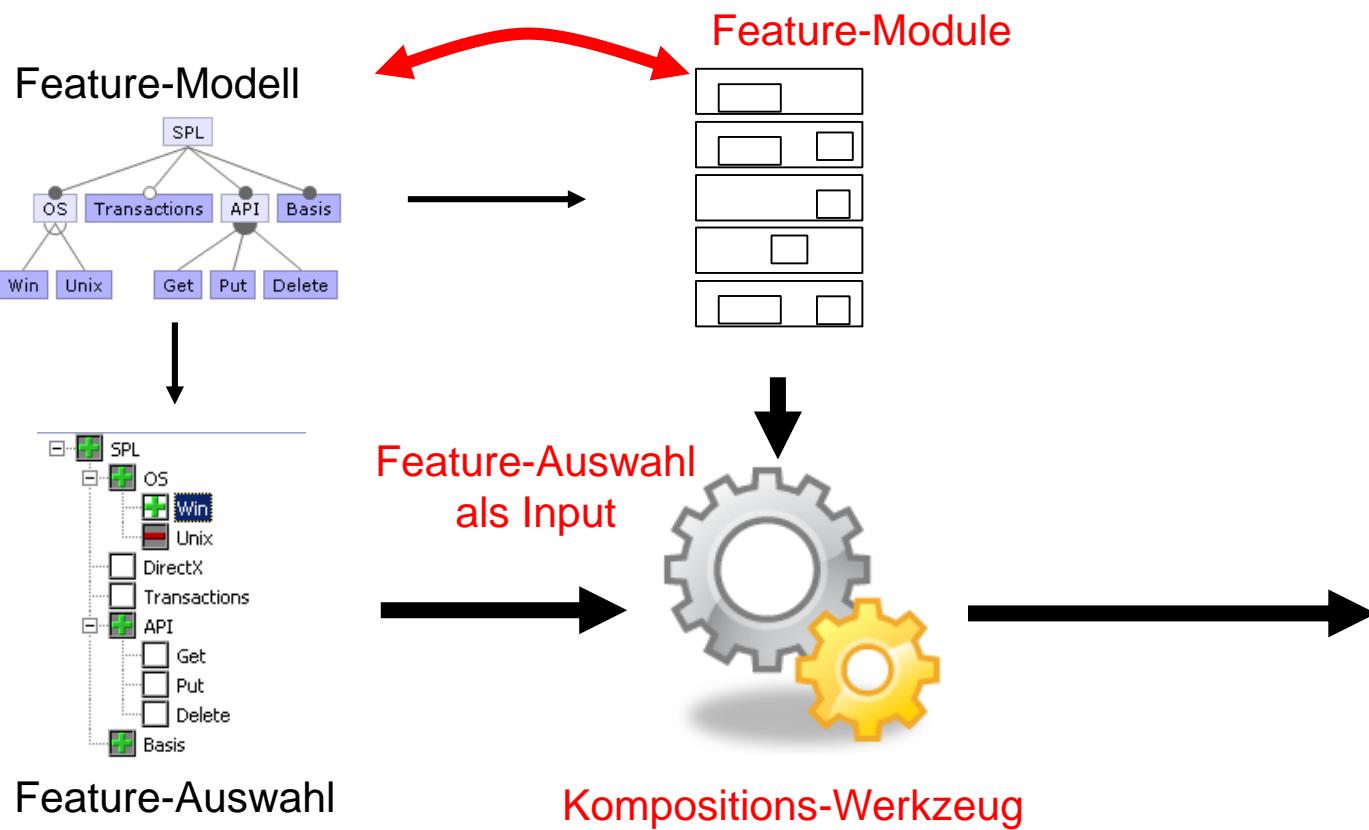


Feature-Komposition



Konfigurierbare Software mit Feature Modulen

1:1-Mapping von Features zu Feature-Modulen



CUST_NO	CUSTOMER	CONTACT	CONTACT...	PHONE...
1	1,001 Signature...	Dale J. Little	(619) 531	
2	1,002 Dallas Tec...	Olen Brown	(214) 961	
3	1,003 Butte, Griff...	James Butte	(617) 481	
4	1,004 Central Bank...	Elizabeth Brocket	61 211 9	
5	1,005 OT Systems...	Tai Wu	(852) 851	
6	1,006 DataServe...	Tomas Bright	(613) 221	
7	1,007 MW Belter...	Mrs. Beau...		
8	1,008 Amin Vacat...	Leilani Briggs	(800) 831	
9	1,009 Max...	Max	22 01 23	
10	1,010 M&M Corp...	Mauricio M...	5 99 27	

Fertiges Programm

Implementierung mittels AHEAD

Wie können Feature-Module implementiert werden?

- Trennung in mehrere Klassen ist bewährt und eine geeignete Grundstruktur
- Features werden oft in mehreren Klassen implementiert
- Klassen implementieren mehr als ein Feature
- Grundlegende Idee: Klassenstruktur behalten und basierend auf Features weiter aufteilen
- AHEAD (Algebraic Hierarchical Equations for Application Design) oder FeatureHouse als Ansätze

Dekomposition von Klassen

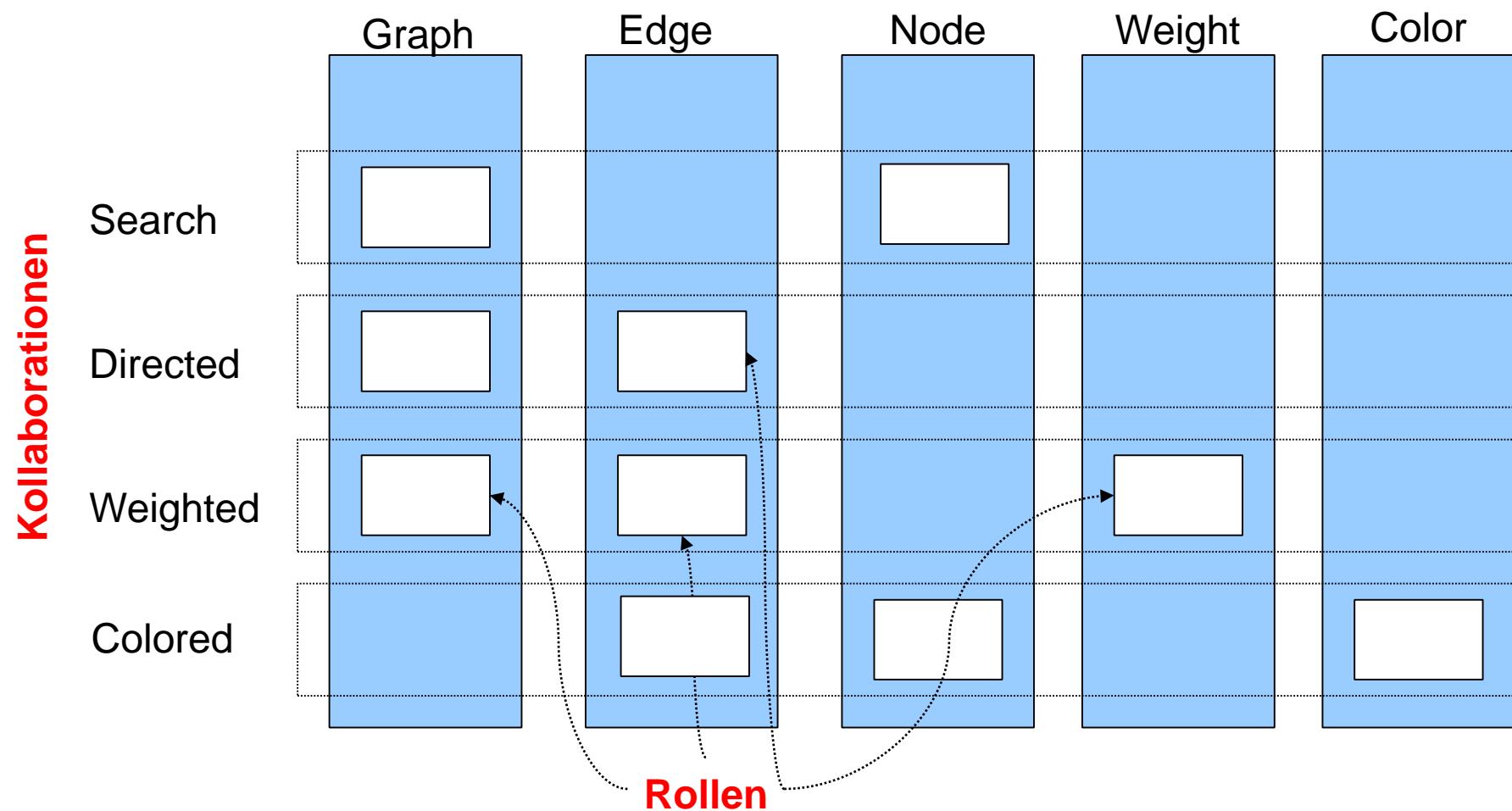
	Klasses				
	Graph	Edge	Node	Weight	Color
Search	■	■	■	■	■
Directed	■	■	■	■	■
Weighted	■	■	■	■	■
Colored	■	■	■	■	■

Kollaborationen & Rollen

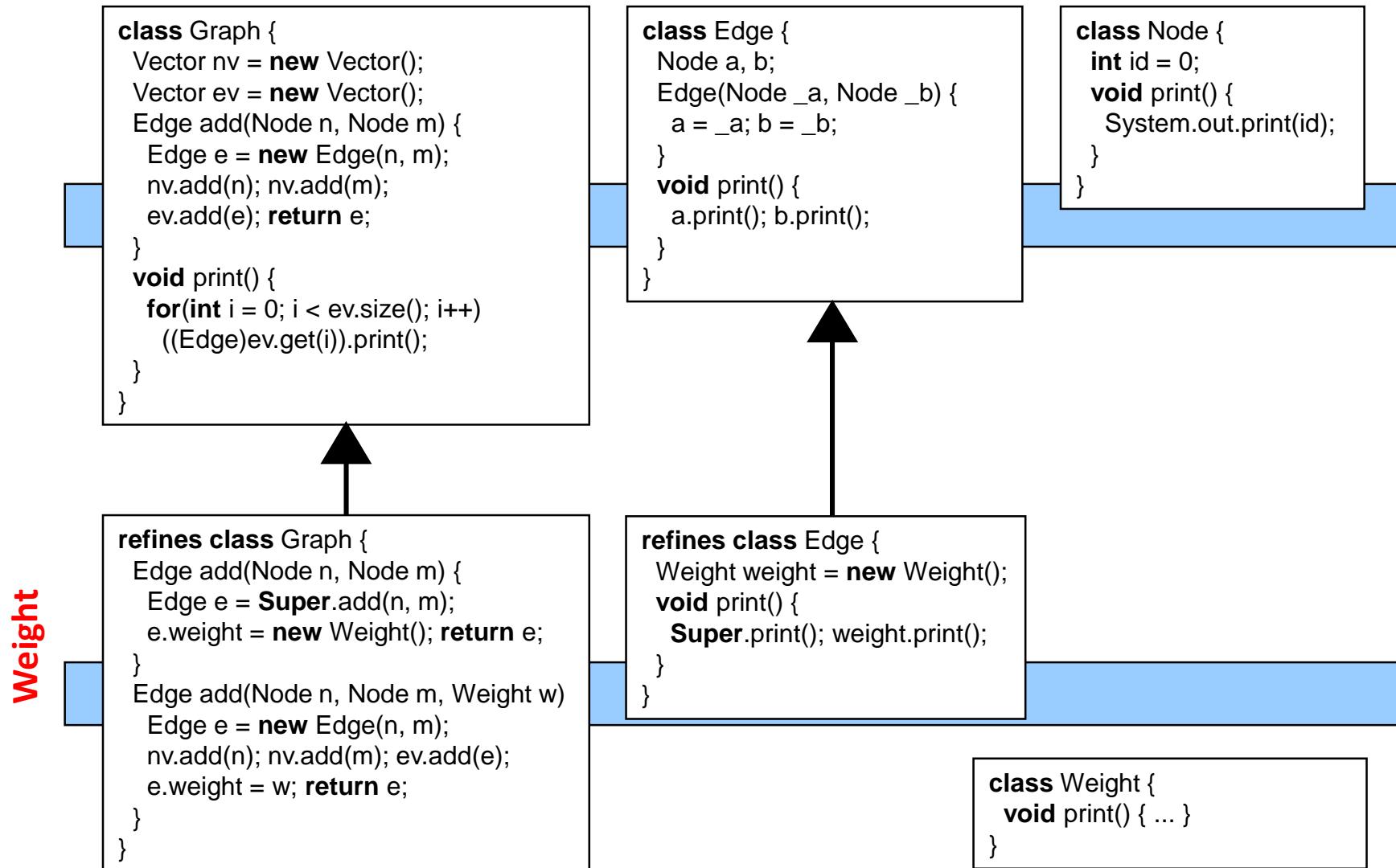
- **Kollaboration:** Eine Menge von Klassen, die miteinander interagieren, um ein Feature zu implementieren
- Verschiedene Klassen spielen verschiedene Rollen in einer Kollaboration
- Eine Klasse spielt verschiedene Rollen in verschiedenen Kollaborationen
- Eine Rolle kapselt das Verhalten/die Funktionalität einer Klasse, die für eine Kollaboration gebraucht wird

Kollaborationen & Rollen

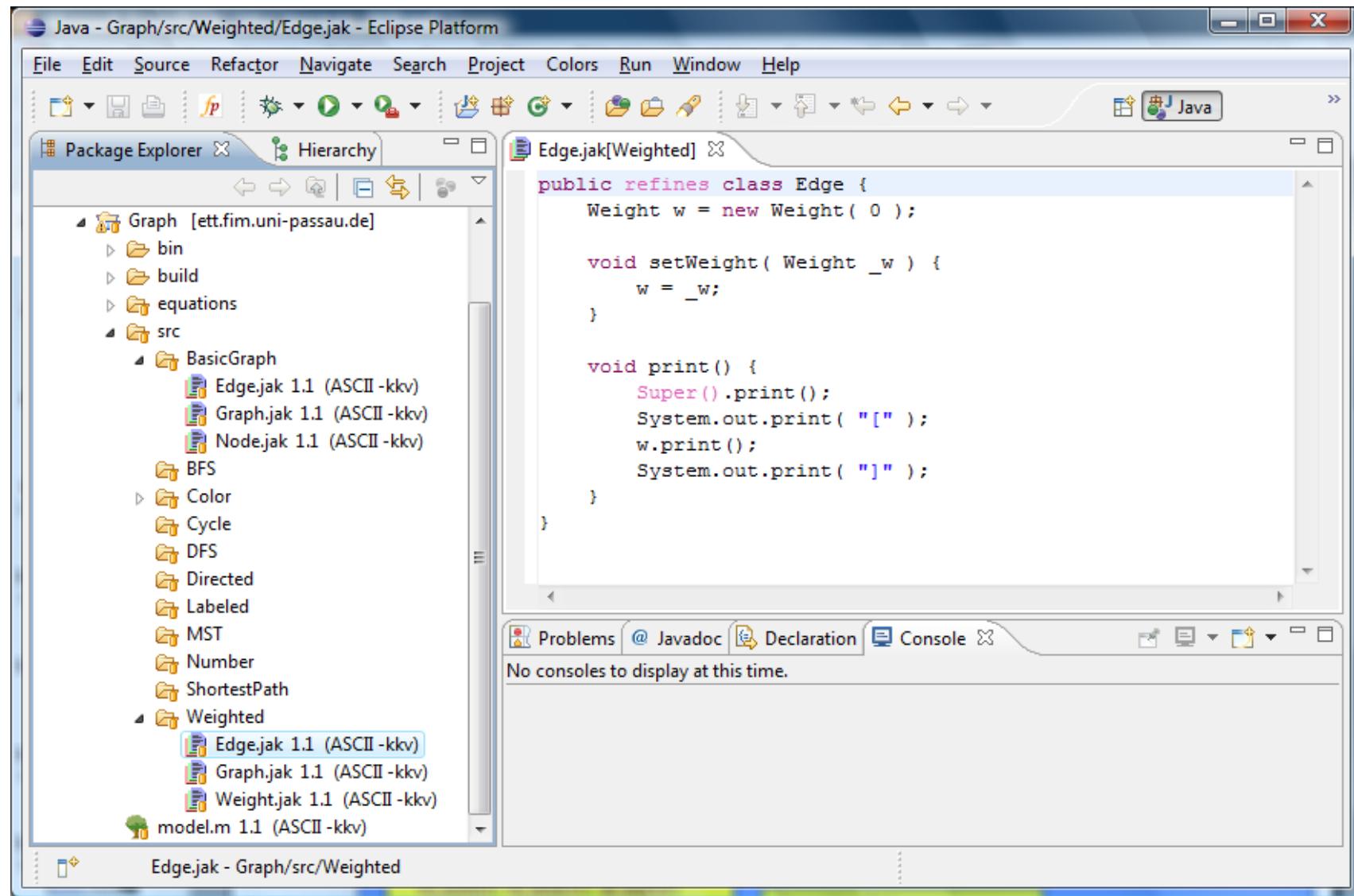
Classes



Kollaboration: Design



Hierarchie im Dateisystem



Beispiel: Verfeinerung von Klassen

Schrittweise Verfeinerung der Basis-Implementierung mittels Extensions

Edge.jak

```
class Edge {  
    ...  
}
```

Edge.jak

```
refines class Edge {  
    private Node start;  
    ...  
}
```

Edge.jak

```
refines class Edge {  
    private int weight;  
    ...  
}
```

Methoden-Verfeinerung (AHEAD)

- Methoden können in jeder Verfeinerung hinzugefügt oder erweitert werden
- Methoden können überschrieben werden (overriding)
- Die Methode der vorigen Verfeinerung wird mittels **Super** aufgerufen
- Ähnlich zu Vererbung

```
class Edge {  
    void print() {  
        System.out.print(  
            " Edge between " + node1 +  
            " and " + node2);  
    }  
}
```

```
refines class Edge {  
    private Node start;  
    void print() {  
        Super().print();  
        System.out.print(  
            " directed from " + start);  
    }  
}
```

```
refines class Edge {  
    private int weight;  
    void print() {  
        Super().print();  
        System.out.print(  
            " weighted with " + weight);  
    }  
}
```

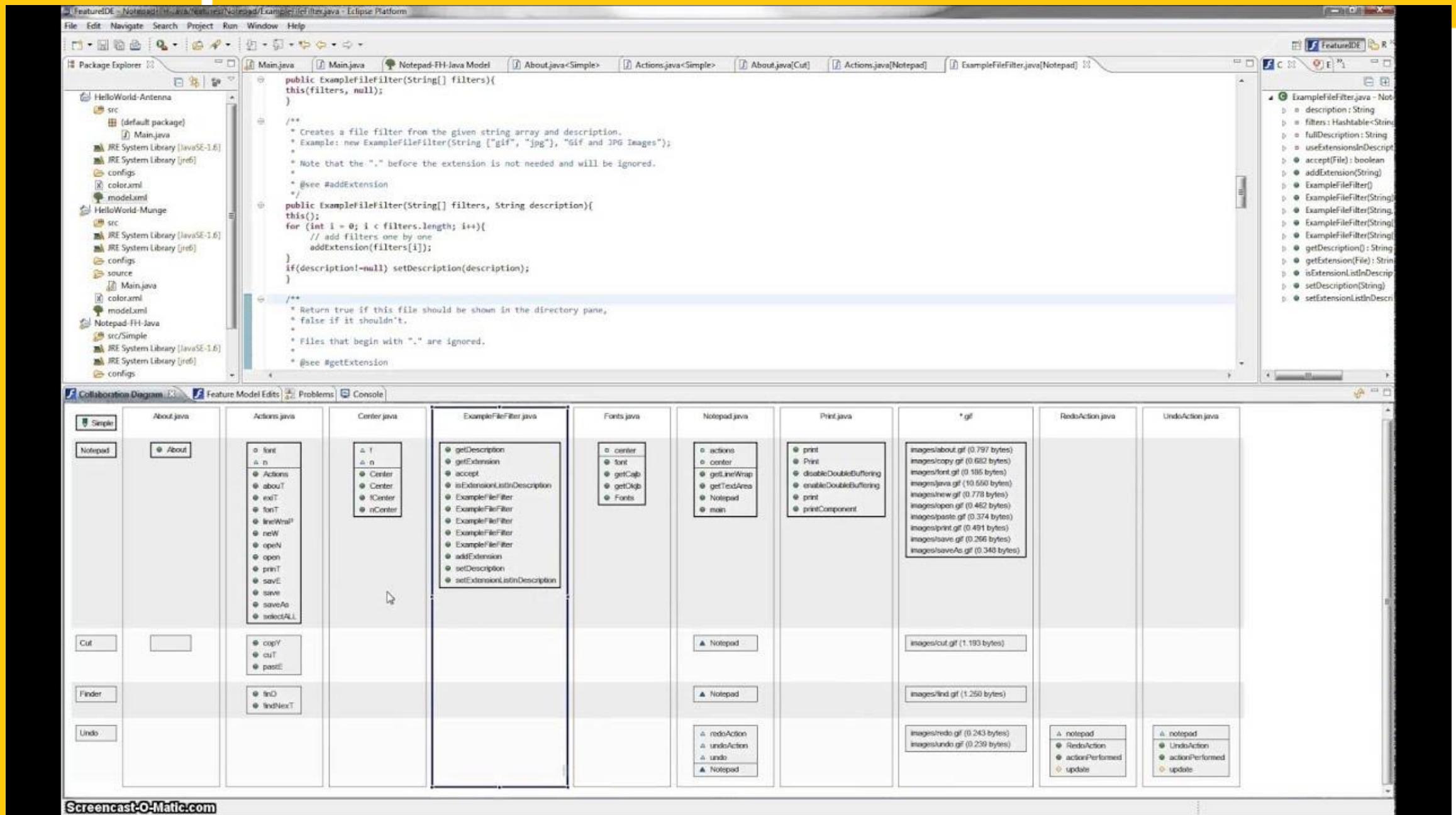
Methoden-Verfeinerung (FeatureHouse)

- **refines** nicht notwendig
- Methoden können in jeder Verfeinerung hinzugefügt oder erweitert werden
- Method-Overriding
- Die Methode der vorigen Verfeinerung wird mittels **original** aufgerufen
- Ähnlich zu Vererbung

```
class Edge {  
    void print() {  
        System.out.print(  
            " Edge between " + node1 +  
            " and " + node2);  
    }  
}
```

```
class Edge {  
    private Node start;  
    void print() {  
        original();  
        System.out.print(  
            " directed from " + start);  
    }  
}
```

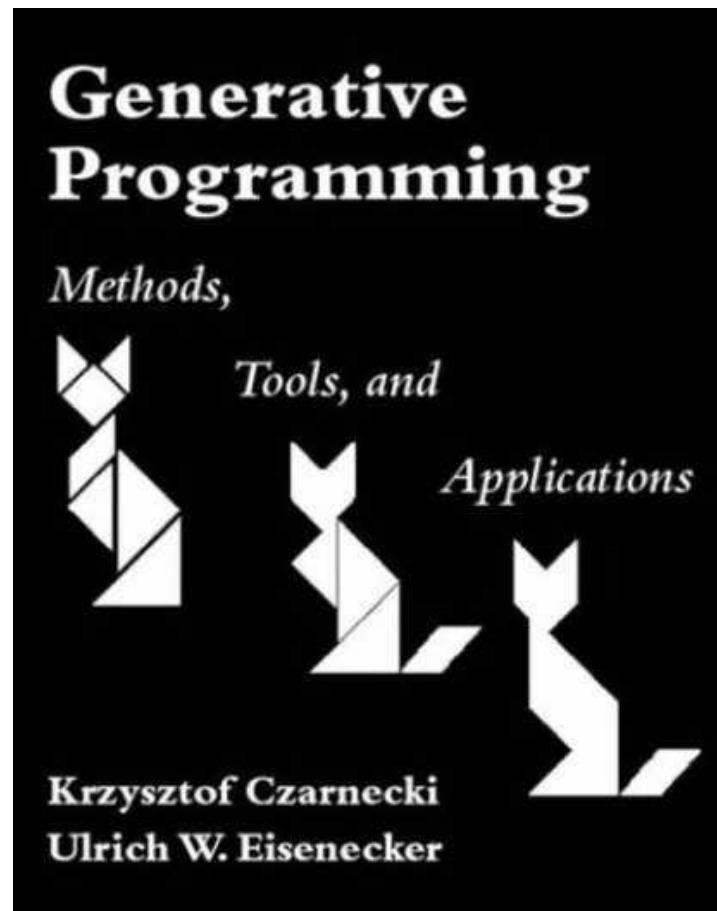
```
class Edge {  
    private int weight;  
    void print() {  
        original();  
        System.out.print(  
            " weighted with " + weight);  
    }  
}
```



Dilemma

- Es ist nicht immer möglich, alle Belange zu modularisieren
- Ein Grundmaß an verstreutem und vermischttem Code in OOP-Implementierungen ist normal
- Einige Belange sind immer „orthogonal“ zu anderen: querschneidende Belange

A "Must Read"



Was Sie mitgenommen haben sollten:

- Nennen/Erklären Sie X Probleme variantenreicher Software. Skizzieren Sie mögliche Lösungen.
- Was sind querschneidende Belange? Was ist das Feature Traceability Problem?
- Erläutern Sie die Begriffe Scattering, Tangling, Tyrannei der dominanten Dekomposition.
- Nennen/Erläutern Sie X Vorteile/Nachteile von Präprozessoren.
- Schreiben Sie einen einfachen Taschenrechner. Eingaben für den Taschenrechner werden als Kommandozeilenparameter übergeben. Als Basis-Feature soll die Addition implementiert sein. Implementieren Sie Subtraktion und Multiplikation als optionale Feature jeweils mit den folgenden Techniken: Parameter und Präprozessoren.

Literatur I

- K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, SEI, 1990.
[Frühe Ideen zur Domänenanalyse mit Feature-Modellen]
- K. Czarnecki and U. Eisenecker. Generative Programming: Methods, Tools, and Applications. Addison-Wesley, 2000.
- D. Batory. Feature Models, Grammars, and Propositional Formulas, In Proc. of Software Product Line Conference (SPLC), 2005
- Allgemeine Bücher zu Produktlinien:
 - Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. Feature-Oriented Software Product Lines: Concepts and Implementation. Springer-Verlag, 2013. 308 pages, ISBN 978-3-642-37520-0, text book to appear May 31, 2013.
 - P. Clements, L. Northrop, Software Product Lines : Practices and Patterns, Addison-Wesley, 2002
 - K. Pohl, G. Böckle, F. van der Linden, Software Product Line Engineering: Foundations, Principles, and Techniques, Springer, 2005

Literatur II

- D. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling Step-Wise Refinement. *IEEE Transactions on Software Engineering*, 30(6), 2004.
[Vorstellung von AHEAD]
- S. Apel, C. Kästner, and C. Lengauer. FeatureHouse: Language-Independent, Automated Software Composition. In Proc. Int'l Conf. Software Engineering, 2009.
[Überblick über FSTs und FeatureHouse]
- S. Apel, C. Lengauer, B. Möller, and C. Kästner. An Algebraic Foundation for Automatic Feature-Based Program Synthesis. *Science of Computer Programming*, 2010.
[Formalisierung & Feature-Algebra]