

# Vorlesung "Softwaretechnik"

## Entwurf: Modularisierung

Lutz Prechelt

Freie Universität Berlin, Institut für Informatik

- Modularisierung
  - Modulbegriff
  - Kriterien für Aufteilung
- Fallstudie: KWIC
  - KWIC 1: Datenflussskette
  - Einschätzen der Entwurfsqualität
  - KWIC 2: Zentrale Steuerung
  - KWIC 3: Datenabstraktion
  - Verhalten bei Änderungen
  - Verwandtschaft mit Architekturstilen

# Wo sind wir?: Taxonomie "Die Welt der Softwaretechnik"

## Welt der Problemstellungen:

- Produkt (Komplexitätsprob.)
  - Anforderungen (Problemraum)
  - **Entwurf (Lösungsraum)**
- Prozess (psycho-soziale P.)
  - Kognitive Beschränkungen
  - Mängel der Urteilskraft
  - Kommunikation, Koordination
  - Gruppendynamik
  - Verborgene Ziele
  - Fehler

## Welt der Lösungsansätze:

- Technische Ansätze ("hart")
  - **Abstraktion**
  - Wiederverwendung
  - Automatisierung
- **Methodische Ansätze ("weich")**
  - **Anforderungsermittlung**
  - **Entwurf**
  - **Qualitätssicherung**
  - **Projektmanagement**

- Einsicht: Man sollte *vor* dem Kodieren über eine günstige Struktur der Software nachdenken
  - und diese als Koordinationsgrundlage schriftlich festhalten
- Prinzipien:
  - **Trennung von Belangen**
  - **Architektur**: Globale Struktur festlegen (Grobentwurf), insbes. für das Erreichen der nichtfunktionalen Anforderungen
  - **Modularisierung**: Trennung von Belangen durch Modularisierung, Kombination der Teile durch Schnittstellen (information hiding, Lokalität)
  - **Wiederverwendung**: Erfinde Architekturen und Entwurfsmuster nicht immer wieder neu
  - **Dokumentation**: Halte sowohl Schnittstellen als auch zu Grunde liegende Entwurfsentscheidungen und deren Begründungen fest

# Was wir schon wissen:

1. Eine Architektur beschreibt,
  - welche **Teile** ein System hat,
  - wie diese **zusammenspielen**
  - und wie dadurch die **funktionalen** und **nichtfunktionalen Anforderungen** erfüllt werden können
2. Wichtigster Aspekt ist die Zerlegung in Module:
  - Ein Modul braucht eine klare **Zuständigkeit** und wird beschrieben über seine **Schnittstelle**
  - Die Schnittstelle verbirgt ein oder mehrere **Geheimnisse** des Moduls
  - Die Geheimnisse spiegeln **Entwurfsentscheidungen** wider, die sich ändern könnten
3. Ein gut modularisiertes System ist einfach zu verstehen und Änderungen beschränken sich meist auf ein Modul

# Wie findet man eine gute Modularisierung?

- Es gibt nirgends eine wirkliche Anleitung, wie man das anstellt
- Sondern nur bekannte Kriterien zur Beurteilung des Resultats
- Deshalb studieren wir das Problem an einem Fallbeispiel
  - KWIC – Key Word in Context
- Achtung:
  - Das Problem ist so einfach, dass die verwendete Modularisierung vielleicht stellenweise übertrieben erscheint
  - Bitte stellen Sie sich dann vor, das Problem wäre viel komplexer und übertragen Sie die Überlegungen analog



- Eingabe: Liste von Buchtiteln, z.B.
  - Java Eclipse Manual
  - Eclipse Pocket Guide
  - Definitive NetBeans Guide

- **Problemstellung:** Suche von Büchern in einer Bibliothek nach Titelstichwort – aber ohne Computer, nur auf ausgedruckten Listen
- Ausgabe: "Key Word in Context"-Index
  - Definitive NetBeans Guide
  - Eclipse Manual. Java
  - Eclipse Pocket Guide
  - Guide. Definitive NetBeans
  - Guide. Eclipse Pocket
  - Java Eclipse Manual
  - Manual. Java Eclipse
  - NetBeans Guide. Definitive
  - Pocket Guide. Eclipse

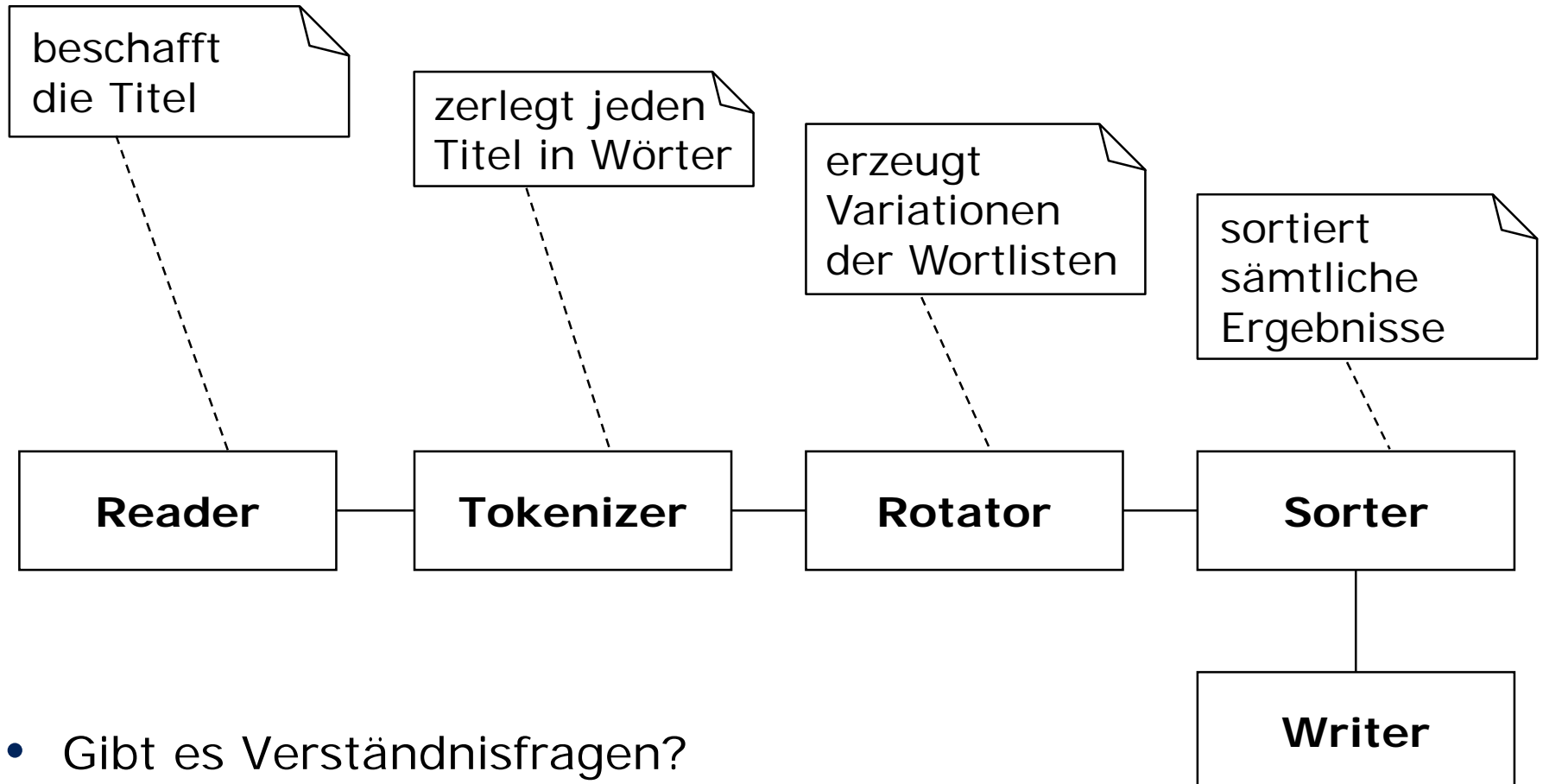
# KWIC: Beispiel (realistische Version)

FEDERAL GOVERNMENT.= THE	FACE-TO-FACE WAR INFORMATION SERVICE OF TH	LAVES WH43
ARCHIVES.= RESEARCH	TWO	RACHRA P 62
POLITICAL SCIENCE ( APSA REPORT ).=	FACILITIES AND MATERIALS AT THE NATIONAL A	GROVER WC40
C STUDY IN POLITICAL SCIENCE ).=	FACILITIES FOR PUBLICATION IN THE FIELD OF	FAIRLI JA30
AMERICAN INDIVIDUALISM-- FACT AND	FACT AND FICTION IN GOVERNMENT ( SCIENTIFI	LOEB 1 34
N SOCIALIST PARTY-- A CASE STUDY IN	FICTION.=	MASON AT52
OTE IN CHICAGO.= A	FACTIONAL CONFLICT.= THE ITALIA	ZARISK R 62
ERSHIP.= AGE AS A	FACTOR ANALYSIS OF THE 1932 PRESIDENTIAL V	GOSNEL HF35
THE ECONOMIC	FACTOR IN THE RECRUITMENT OF COMMUNIST LEA	HOLT RT54
LICY, I AND II.= LEGAL AND ECONOMIC	FACTOR IN THE ROOSEVELT ELECTIONS.=	OGRURN WF40
I SOUTH.= SOCIAL AND ECONOMIC	FACTORS AFFECTING SOVIET RUSSIA'S FOREIGN	PRINCE C 44
SOUTH.= POLITICAL	FACTORS AND NEGRO VOTER REGISTRATION IN TH	MATTHE CR63
INTRATERRITORIAL POWERS OF CITIES AS	FACTORS AND NEGRO VOTER REGISTRATION IN TH	MATTHE CR63
SOME NEGLECTED	FACTORS IN CALIFORNIA METROPOLITAN GOVERN	CROUCH WF37
N OF PSYCHOLOGICAL AND SOCIOLOGICAL	FACTORS IN LAW-MAKING.=	BRUNCK E 14
OPERATION, 1945-1950.= POLITICAL	FACTORS IN POLITICAL BEHAVIOR.= INTERACTIO	FRENKE E 52
LINA.= CERTAIN PERSONALITY	FACTORS IN U.S. INTERNATIONAL FINANCIAL CO	BEHRMA JN53
ION OF METHODS FOR ASCERTAINING THE	FACTORS OF STATE LEGISLATORS IN SOUTH CARD	MCCONA JB50
THE LAW AND THE	FACTORS THAT INFLUENCE JUDICIAL DECISIONS	HALL AR26
OF THE MINNESOTA LEGISLATURE.= THE	FACTS ( STUDY OF POLITICAL SCIENCE ).=	WILSON A 11
CONFERENCE.=	FAILURE OF THE FARMER-LABOR PARTY TO CAPTU	NAFTAL A 44
THE 'OPEN SOCIETY' AND ITS	FAILURES AND SUCCESSES AT THE SECOND HAGUE	REINSC P508
TATE ADMINISTRATIVE STRUCTURE-- THE	FALLACIES.=	KENDAL U 60
VARIATIONS ON A	FALLACY OF THE STATISTICAL APPROACH.= EVAL	JACOBS JM2B
N PARLIAMENT I.	FAMILIAR THEME ( APSA PRESIDENTIAL ADDRESS	COEGAR PH5E
	FAMILY VOTING IN FRANCE ( REPRESENTATION )	GOOCH RK2A

- Eingabe: Eine Menge von Buchtiteln
- Erzeugt zu jedem Titel mit  $N$  Wörtern  $N$  Ergebnisse, indem so oft das erste Wort ans Ende gestellt wird, bis jedes Wort einmal vorn gestanden hat
- Diese Ergebnisse ("Varianten") werden für alle Titel gesammelt und die Gesamtmenge sortiert
- Ausgabe: Die sortierte Variantenliste



- Ein plausibler Vorschlag (Pipe-and-Filter-Architektur)



- Gibt es Verständnisfragen?

Für wie gut halten Sie diese Modularisierung?

- **1** Perfekt
- **2** Gut, ohne erhebliche Schwächen
- **3** OK, es ginge besser, aber sie ist voll brauchbar
- **4** Ausreichend: Funktionsfähig, aber sehr ungünstig aufgeteilt
- **5** Unsinnig

Wie sicher sind Sie sich in diesem Urteil?

- **1** Ganz sicher
- **2** Ich glaube, es stimmt
- **4** Ich hoffe, dass es stimmt
- **5** Gar nicht, es ist geraten

Die Module verbergen relevante Entwurfsentscheidungen:

- *Reader*
  - Datenquelle für Titelliste (z.B. Datei, GUI) samt Einleseverfahren
- *Tokenizer*
  - Was ist ein Wort?
- *Rotator*
  - Datenstruktur und Algorithmus zur effiz. Darstellung d. Varianten
  - Behandlung des Titelanfangs im Varianteninnern
- *Sorter*
  - Datenstruktur und Algorithmus zum Sortieren
  - Sortierordnung
- *Writer*
  - Ausgabekanal
  - Formataufbereitung

# "Das ist doch trivial!" ?

- Diese Geheimnisse sind nicht so banal, wie Sie vielleicht denken
- z.B. "Was ist ein Wort" beim *Tokenizer*
  - Vorschläge für die Definition?
- Ist Ihre Definition für folgende Buchtitel angemessen?
  - "Beitrag zur Populationsgenetik der sauren **Erythrocytenphosphatase-acP-EC 3.1.3.2** unter besonderer Berücksichtigung des reinerbigen Typus C" (1980)
  - "Lepton-Hadron-Korrelationen in (2+1)-Jet-Produktion in tiefinelastischer Elektron-Proton-Streuung zur  **$O(\alpha^2 s)$** " (1992)
  - "Die molekulare Wirkung von **2,4,5-** und **2,4,6-Trichlorphenol** auf Eukaryontenzellen" (1990)

- Ein wirklich guter *Tokenizer* würde
  - erkennen, um welches Fachgebiet es sich handelt,
  - gebietsspezifische Regeln für die Wortbegrenzung verwenden
  - und evtl. sogar Wörter anders zusammenbauen!
    - z.B. aus "2,4,5- und 2,4,6-Trichlorphenol"  
mache "2,4,5-Trichlorphenol und 2,4,6-Trichlorphenol"
- Er müsste also mehrere fachspezifische Expertensysteme enthalten!

- Auch die Sortierordnung ist nicht selbstverständlich:
- Wie sortiert man ä æ å ?
  - Für Deutsche: ä=ae æ=ae å=a
  - Für Amerikaner: ä=a æ=ae å=a
  - Für Skandinavier: ä>z æ>z å>z
- Wie sortiert man ß ?
  - Für Deutsche: ß=ss
  - Für Amerikaner?
  - Für Skandinavier?
- Wie sortiert man ð þ ł ŋ ?
  - Alles europäische Buchstaben!
    - Anmerkung: Landesspezifische genormte Regeln sind in Java eingebaut vorhanden
- Wie sortiert man β (beta) ε (epsilon) ?
- Wie sortiert man asiatische Ideogrammschriften?

- Die Ausgabe der Liste kann ganz verschieden sein:
- Im einfachen Fall eine schlichte Textdatei
  - nur die rohen Ergebnisse
- Aber vielleicht wollen wir ja ein schönes PDF?
  - Dann muss das Modul ziemlich viel "wissen". Teilaufgaben:
    - PDF-Dateiformat
    - Seitenumbruch (Seitengrößen, Fontheiten etc.)
    - Kopfzeilen, Fußzeilen, Seitennummern etc.
    - Inhaltsverzeichnis (per Anfangsbuchstabe)

Für wie gut halten Sie diese Modularisierung?

- **1** Perfekt
- **2** Gut, ohne erhebliche Schwächen
- **3** OK, es ginge besser, aber sie ist voll brauchbar
- **4** Ausreichend: Funktionsfähig, aber sehr ungünstig aufgeteilt
- **5** Unsinnig

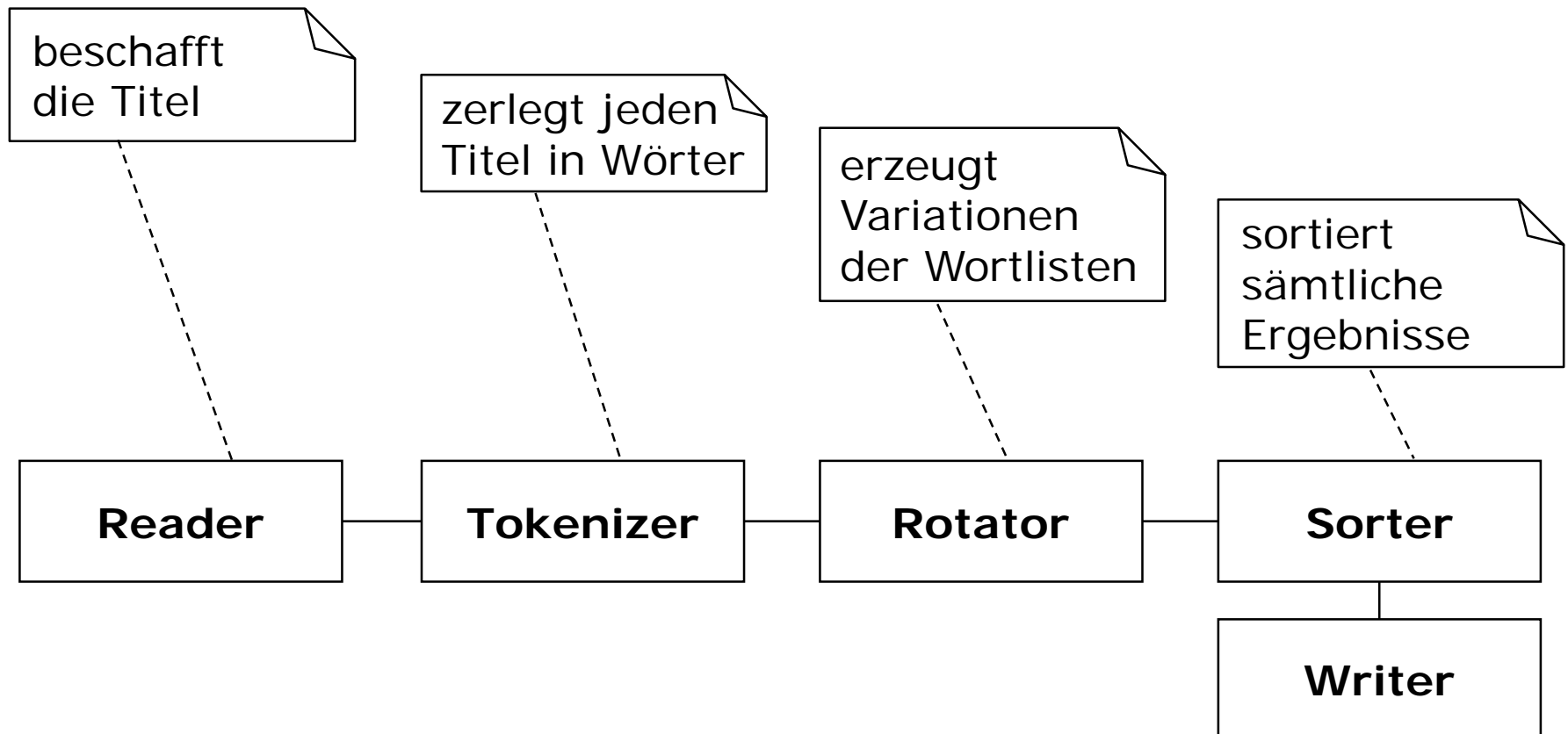
Wie sicher sind Sie sich in diesem Urteil?

- **1** Ganz sicher
- **2** Ich glaube, es stimmt
- **4** Ich hoffe, dass es stimmt
- **5** Gar nicht, es ist geraten

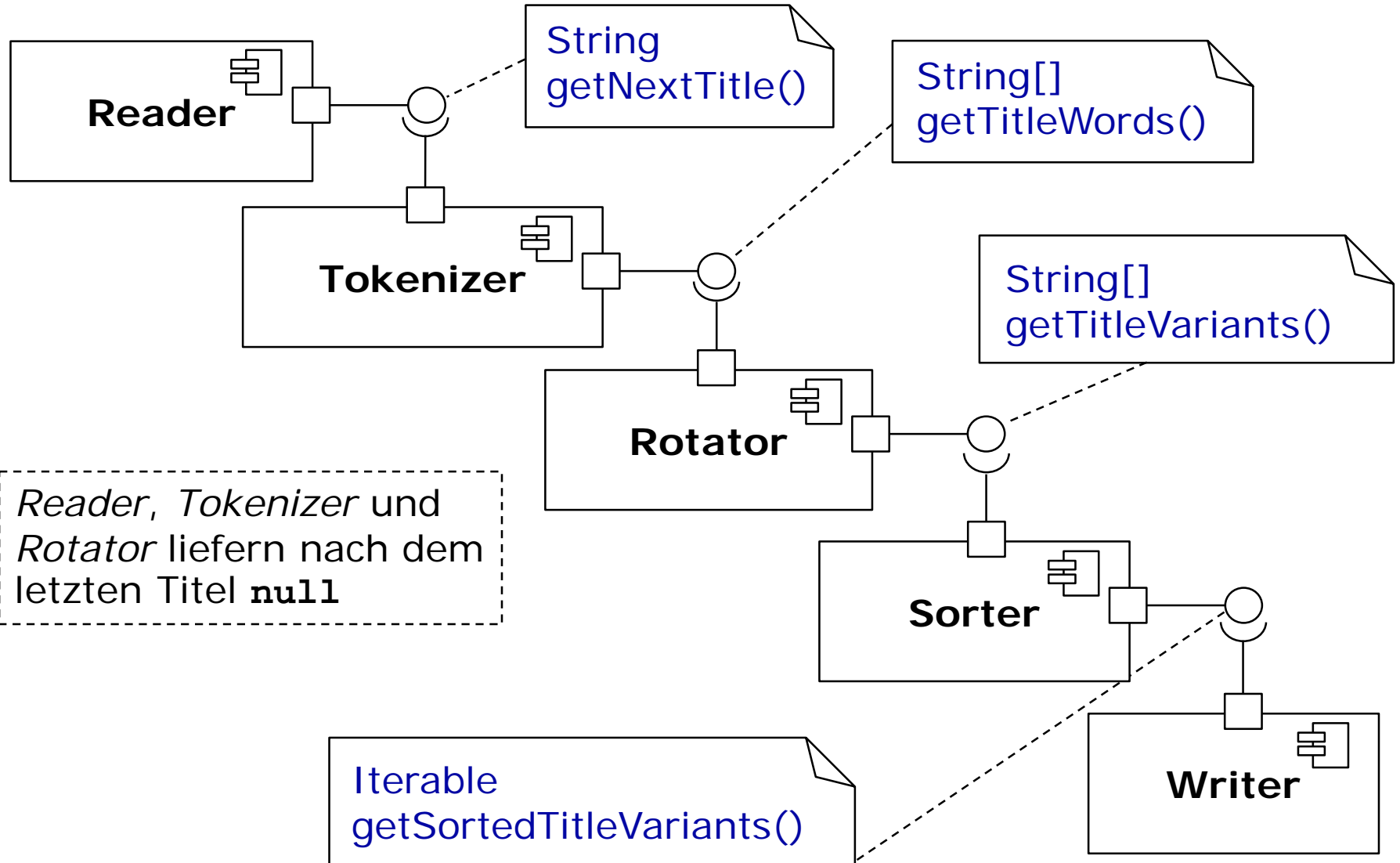


# Erinnerung: KWIC Modularisierung

- Diese Darstellung ist reichlich informell und vage
  - Besser wäre eine richtiges Komponentendiagramm
  - mit Angabe der Schnittstellen



# KWIC als UML-Komponentendiagramm



- Das Hauptprogramm (steht in *Writer*) sieht bei dieser Modularisierung sinngemäß etwa so aus:

```
Reader reader      = new FileReader(args[0]);  
Tokenizer tokenizer = new Tokenizer(reader);  
Rotator rotator    = new Rotator(tokenizer);  
Sorter sorter      = new Sorter(rotator);  
Writer writer      = new Writer(sorter);  
writer.writeKWIClist();
```

# KWIC: Mängel dieses Entwurfs

1. Die Module sind unnötig fest aneinander gekoppelt
  - Das erkennt man am Hauptprogramm:
  - Wollte man noch einen Zwischenschritt einfügen, müssten bestehende Schnittstellen geändert werden (Konstruktoren)
    - d.h. die Art und Reihenfolge der Schritte ist nicht Geheimnis irgendeines Moduls
2. Die Schnittstellen der Module sind hochspezialisiert für genau diese Anwendung
  - Obwohl man z.B. einen *Tokenizer* ja auch woanders wiederverwenden könnte
  - Die Module sollten besser alle von einem Treiber aus aufgerufen werden, anstatt sich gegenseitig als Kette aufzurufen
  - Dann würden die Schnittstellen allgemeiner und die Module wären besser wiederzuverwenden
    - Denn dann wären Art und Reihenfolge der Schritte das Geheimnis dieses Treibermoduls

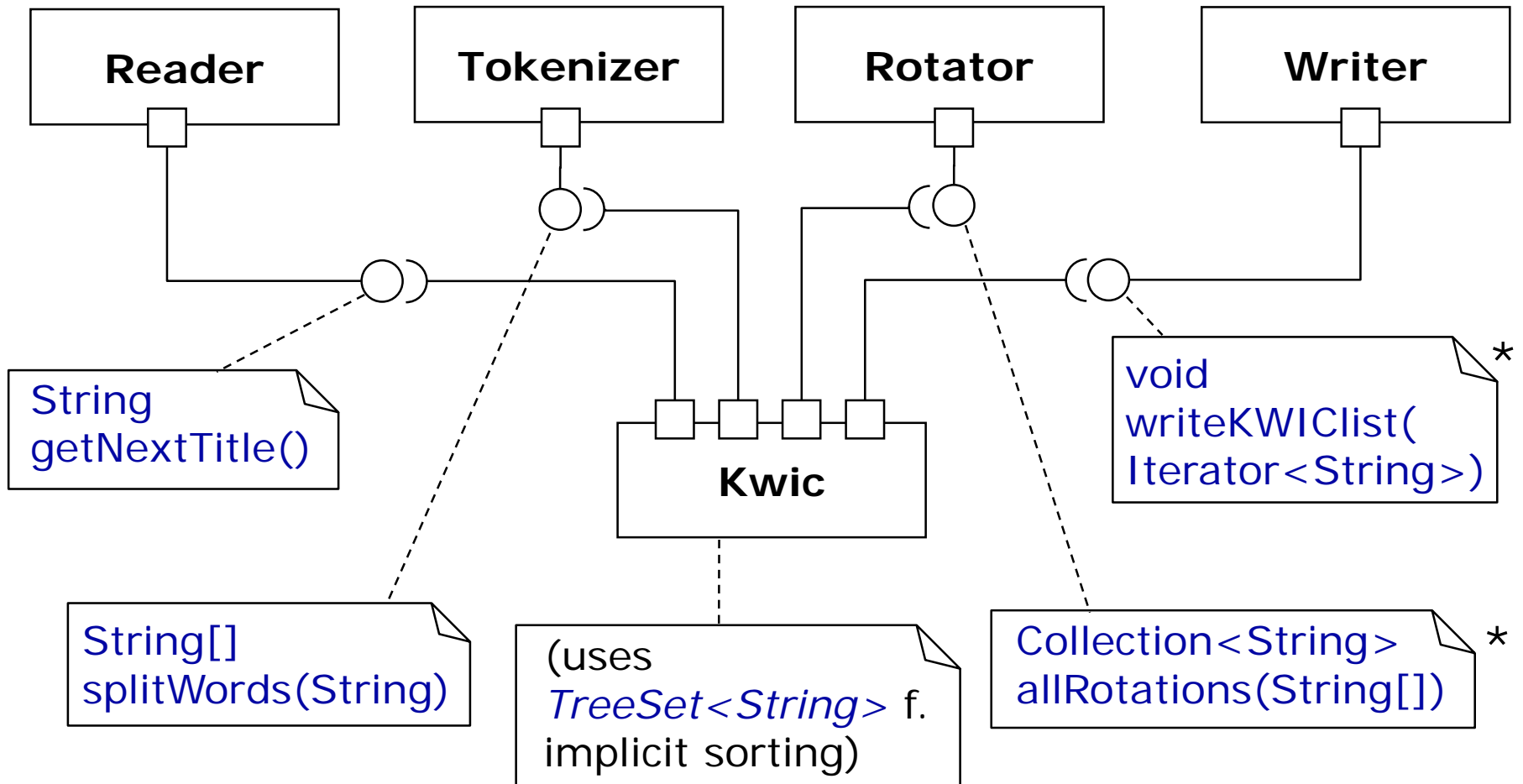
Für wie gut halten Sie diese Modularisierung?

- **1** Perfekt
- **2** Gut, ohne erhebliche Schwächen
- **3** OK, es ginge besser, aber sie ist voll brauchbar
- **4** Ausreichend: Funktionsfähig, aber sehr ungünstig aufgeteilt
- **5** Unsinnig

Wie sicher sind Sie sich in diesem Urteil?

- **1** Ganz sicher
- **2** Ich glaube, es stimmt
- **4** Ich hoffe, dass es stimmt
- **5** Gar nicht, es ist geraten

# KWIC Version 2: Mit Treiber



Natürlich gilt auch hier:

- Die Signatur ist nicht die Schnittstelle!
- Insbesondere muss genau der Inhalt der *Collection* und des *Iterators* definiert werden
  - In Java sind sie als *Collection<String>* und *Iterator<String>* definiert – das hilft schon mal *ein wenig*
  - Aber *allRotations()* sollte z.B. beschreiben:  
"A list of N words will result in N variants as follows:
    - Variant 1 is the original list.
    - Variant i+1 is produced from variant i by removing the first word and appending it to the end."

1. *Tokenizer* und *Rotator* haben jetzt "natürliche" Schnittstellen
  - Die enge Kopplung an den Rest des Systems ist verschwunden
  - Dadurch sind die Module jetzt **wiederverwendbar**
2. Das System wird **offener für Erweiterungen**
  - Einfach in Modul *Kwic* ergänzen
3. *Sorter* ist **nicht mehr nötig**
  - Statt dessen benutzt *Kwic* direkt die entsprechenden Klassen aus der Bibliothek:
    - `java.util.TreeSet` sammelt Varianten und sortiert sie
    - `java.text.RuleBasedCollator` legt eine Sortierreihenfolge fest



## KWIC 2: Hauptprogramm

- Das Hauptprogramm (in Klasse *Kwic*) sieht bei dieser Modularisierung sinngemäß etwa so aus:

```

Reader reader          = new Reader(args[0]);
Tokenizer tokenizer    = new Tokenizer();
Rotator rotator       = new Rotator();
Writer writer         = new Writer();
Collator ordering     = Collator.getInstance(); // default sort rules
ordering.setStrength(Collator.SECONDARY); // ignore case
TreeSet<String> kwic = new TreeSet<String>(ordering);
for(;;) { // for each title:
    String title = reader.getNextTitle();
    if (title == null) break; // end of input
    String[] tokens = tokenizer.splitWords(title);
    Collection<String> variants = rotator.allRotations(tokens);
    kwic.addAll(variants);
}
writer.writeKWIClist(kwic.iterator());

```

Für wie gut halten Sie diese Modularisierung?

- **1** Perfekt
- **2** Gut, ohne erhebliche Schwächen
- **3** OK, es ginge besser, aber sie ist voll brauchbar
- **4** Ausreichend: Funktionsfähig, aber sehr ungünstig aufgeteilt
- **5** Unsinnig

Wie sicher sind Sie sich in diesem Urteil?

- **1** Ganz sicher
- **2** Ich glaube, es stimmt
- **4** Ich hoffe, dass es stimmt
- **5** Gar nicht, es ist geraten

- Dieser Entwurf ist gut, solange der komplette KWIC-Index bequem in den Hauptspeicher passt
- Für diese Anwendung (KWIC) ist das akzeptabel, aber in anderen Fällen wäre es gut, eine Architektur zu haben, die
  - entweder den Speicheraufwand stark reduziert
  - oder erlaubt, die Daten im Hintergrundspeicher zu halten (Platte)

## KWIC 2: Speicherbedarf

- KWIC-Listen enthalten offensichtlich sehr viel Redundanz
  - immer wieder die gleichen Wörter
  - immer wieder die gleichen Wortfolgen (nur verschoben)
- Bei langen Titeln wird der Speicherbedarf recht groß:
  - *"Untersuchungen über Supravitalfärbungen an Bullenspermien im Vergleich zu deren Bewegungsverhalten im Nativpräparat unter besonderer Berücksichtigung verschiedener Samenverdünnungen und -aufbewahrungen"* (1959)
  - *"Ermittlung und Wichtung der Merkmale zur Analyse und zum Vergleich von Milchproduktionsanlagen in Zusammenarbeit mit 15 Leitern der Tierproduktion von landwirtschaftlichen Betrieben der Deutschen Demokratischen Republik (VEG und LPG)"* (1972)
    - 29 Wörter. Angenommen, wir hätten 1 Million solcher Titel:
  - ohne Redundanz: 230 Zeichen Rohdaten → 230 MB
  - mit Redundanz: 6670 Zeichen Variantendaten → 6,7 GB

Für wie gut halten Sie diese Modularisierung?

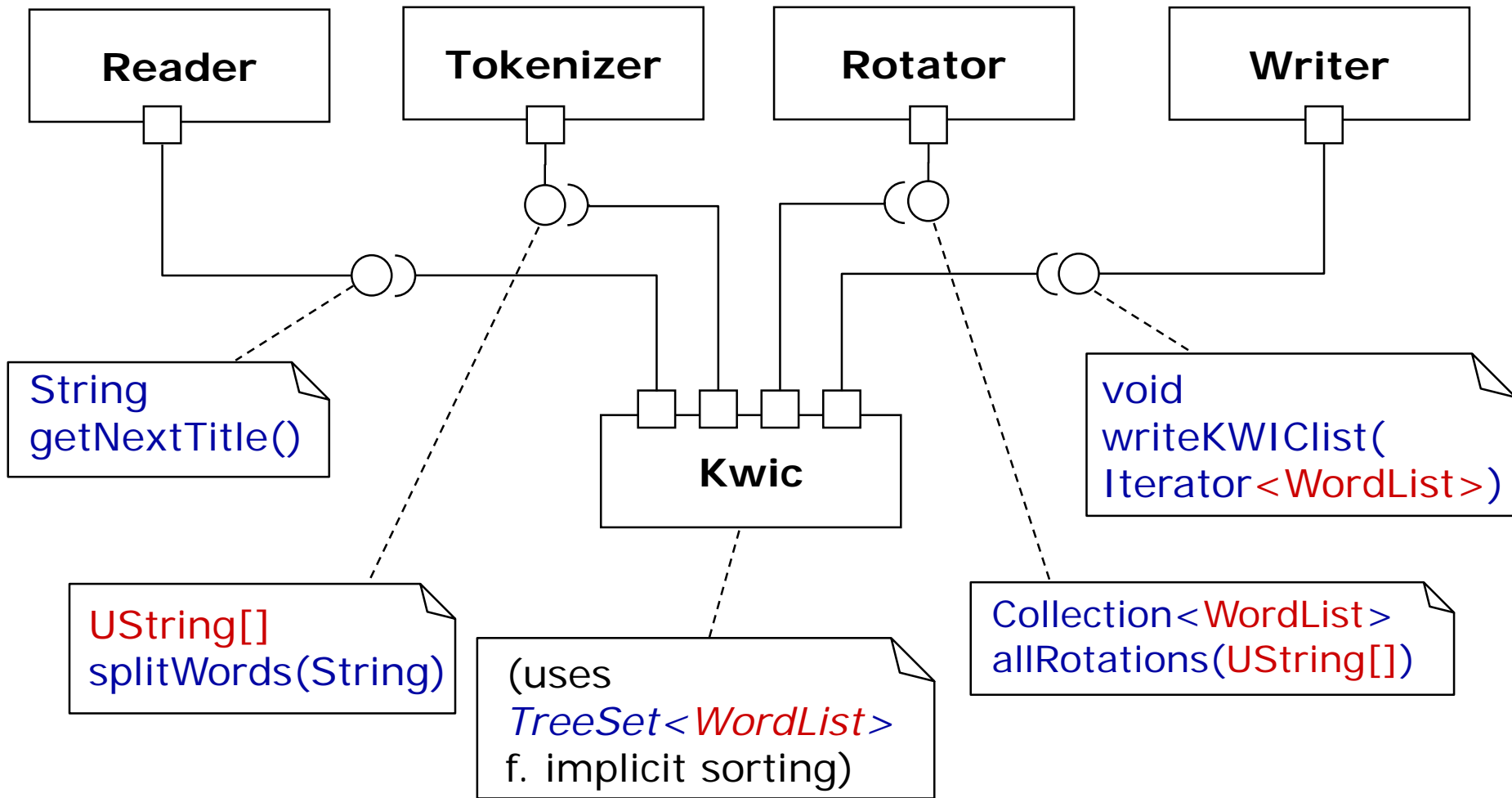
- **1** Perfekt
- **2** Gut, ohne erhebliche Schwächen
- **3** OK, es ginge besser, aber sie ist voll brauchbar
- **4** Ausreichend: Funktionsfähig, aber sehr ungünstig aufgeteilt
- **5** Unsinnig

Wie sicher sind Sie sich in diesem Urteil?

- **1** Ganz sicher
- **2** Ich glaube, es stimmt
- **4** Ich hoffe, dass es stimmt
- **5** Gar nicht, es ist geraten

- Speicherersparnis gelingt, wenn man anstelle von *String* für Wörter und Varianten entsprechende abstrakte Datentypen benutzt:
  - *UString*: Gleiche Strings werden nur einmal angelegt
  - *WordList*: Varianten eines Titels teilen sich ein *UString*-Array
- Durch Einführung von *UString* und *WordList* ändert sich am restlichen Programm nur wenig
  - meist müssen nur Typnamen ersetzt werden
- aber wir können nun zwei Geheimnisse nach *WordList* verlegen
  - Sortierreihenfolge: `static int compare (WordList, WordList)`
  - Format f. Variante: `String toString()`
- und der Speicherbedarf für große KWIC-Listen sinkt stark

# KWIC 3: Mit UString und WordList



Module *UString* und *WordList* sind nicht separat dargestellt

## KWIC 3: Hauptprogramm

- Das Hauptprogramm (in Klasse *Kwic*) sieht bei dieser Modularisierung etwa so aus (Änderungen in rot):

```

Reader reader          = new Reader(args[0]);
Tokenizer tokenizer    = new Tokenizer();
Rotator rotator       = new Rotator();
Writer writer         = new Writer();
TreeSet<WordList> kwic = new TreeSet<WordList>(
    WordList.getComparator());
for (;;) {             // for each title:
    String title = reader.getNextTitle();
    if (title == null) break; // end of input
    UString[] tokens = tokenizer.splitWords(title);
    Collection<WordList> variants = rotator.allRotations(tokens);
    kwic.addAll(variants);
}
writer.writeKWIClist(kwic.iterator());

```



Für wie gut halten Sie diese Modularisierung?

- **1** Perfekt
- **2** Gut, ohne erhebliche Schwächen
- **3** OK, es ginge besser, aber sie ist voll brauchbar
- **4** Ausreichend: Funktionsfähig, aber sehr ungünstig aufgeteilt
- **5** Unsinnig

Wie sicher sind Sie sich in diesem Urteil?

- **1** Ganz sicher
- **2** Ich glaube, es stimmt
- **4** Ich hoffe, dass es stimmt
- **5** Gar nicht, es ist geraten

- Jetzt untersuchen wir das Verhalten der drei Entwürfe bei Änderungen am System
- Änderung 1: Stoppwortliste
- Änderung 2: Wortumbruch
- Änderung 3: Automatisches Abkürzen
- Änderung 4: Datenhaltung auf Hintergrundspeicher

# KWIC: Einführung einer Stoppwortliste

- Es gibt in Titeln viele nichtssagende Wörter
  - Artikel ("der", "ein", ...), Konjunktionen ("und", "weil", ...), Pronomen ("die", "dessen", ...)
- KWIC-Einträge, die mit einem solchen "Stoppwort" beginnen, sollen unterdrückt werden
  - Die Stoppwortliste ist das Geheimnis eines eigenen Moduls
- Die Änderung sieht in allen drei KWIC-Versionen gleich aus:
  - Realisierung eines Stoppworttests in einem Extramodul
  - Aufruf des Tests im *Rotator*
    - Oder bei KWIC 1: zur Erhaltung der pipe-and-filter-Architektur als eigene Filterstufe realisiert

- Lange Titel passen in der Ausgabe nicht auf eine Zeile
- *Writer* soll einen geeigneten Wortumbruch vornehmen
  - Das genaue Verfahren wird sein zusätzliches Geheimnis
    - Man könnte auch ein separates Modul dafür einführen
- KWIC 1, KWIC 2:
  - *Writer* muss dazu zuerst die fertig formatierten Variantenstrings wieder in Wörter zerlegen
  - Das ist ungünstig, denn wir haben ja zuvor mal Wörter gehabt
- KWIC 3:
  - Die Varianten kommen als *WordList*-Objekte an
  - Der Wortumbruch kann also bei der Formatierung recht einfach durchgeführt werden

# KWIC: Automatisches Abkürzen

- Es wäre sinnvoll, bestimmte häufige Wörter automatisch bei der KWIC-Erzeugung durch eine Abkürzung zu ersetzen ("Konferenz" → "Konf.", "international" → "int'l.", ...)
- Dafür soll ein Abkürzmodul *Compactor* ergänzt werden
  - Aufruf zwischen *Tokenizer* und *Rotator*
- KWIC 1:
  - Schnittstelle von *Rotator* ändert sich
  - Hauptprogramm ändert sich
- KWIC 2, KWIC 3:
  - Alle bestehenden Modulschnittstellen bleiben gleich
  - Nur Hauptprogramm ändert sich

- Bei riesigen Mengen von Titeln kann der KWIC-Index nicht mehr im Hauptspeicher gehalten werden
  - selbst mit KWIC 3 vielleicht nicht
- Die ganze Datenhaltung der Titelwörter soll deshalb in einem Datenbankmanagementsystem (DBMS) erfolgen
- KWIC 1:
  - Totale Umstrukturierung von *Rotator*, *Sorter* und *Writer* nötig
    - und vieler weiterer Module, sollten sie existieren
  - Resultierendes System wird sehr kompliziert
- KWIC 2:
  - Totale Umstrukturierung von *Rotator*, *Writer* und *Kwic*
  - Sortieren??? Resultierendes System wird sehr kompliziert
- KWIC 3:
  - Änderungen sind beschränkt auf *UString* und *WordList*
    - Egal wie viele andere Module existieren

- Die Änderungsfreundlichkeit der drei Modularisierungen im direkten Vergleich:

Änderung	KWIC 1	KWIC 2	KWIC 3
1. Stoppwortliste	(+)	+	+
2. Wortumbruch	-	-	+
3. Autom. Abkürzen	-	+	+
4. Hintergrundspeicher	-	-	+

Für wie gut halten Sie diese Modularisierung?

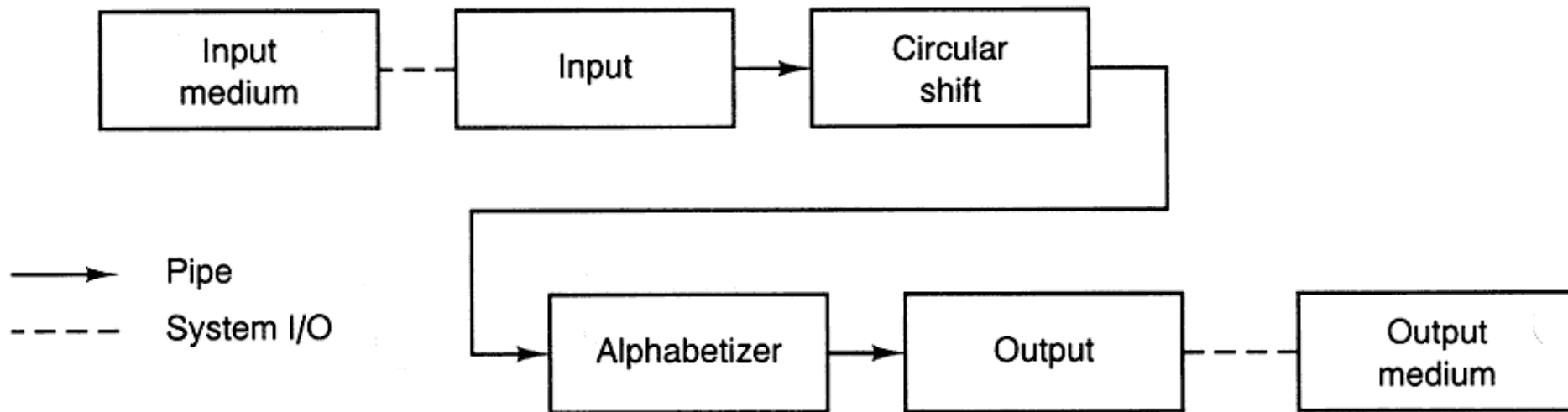
- **1** Perfekt
- **2** Gut, ohne erhebliche Schwächen
- **3** OK, es ginge besser, aber sie ist voll brauchbar
- **4** Ausreichend: Funktionsfähig, aber sehr ungünstig aufgeteilt
- **5** Unsinnig

Wie sicher sind Sie sich in diesem Urteil?

- **1** Ganz sicher
- **2** Ich glaube, es stimmt
- **4** Ich hoffe, dass es stimmt
- **5** Gar nicht, es ist geraten

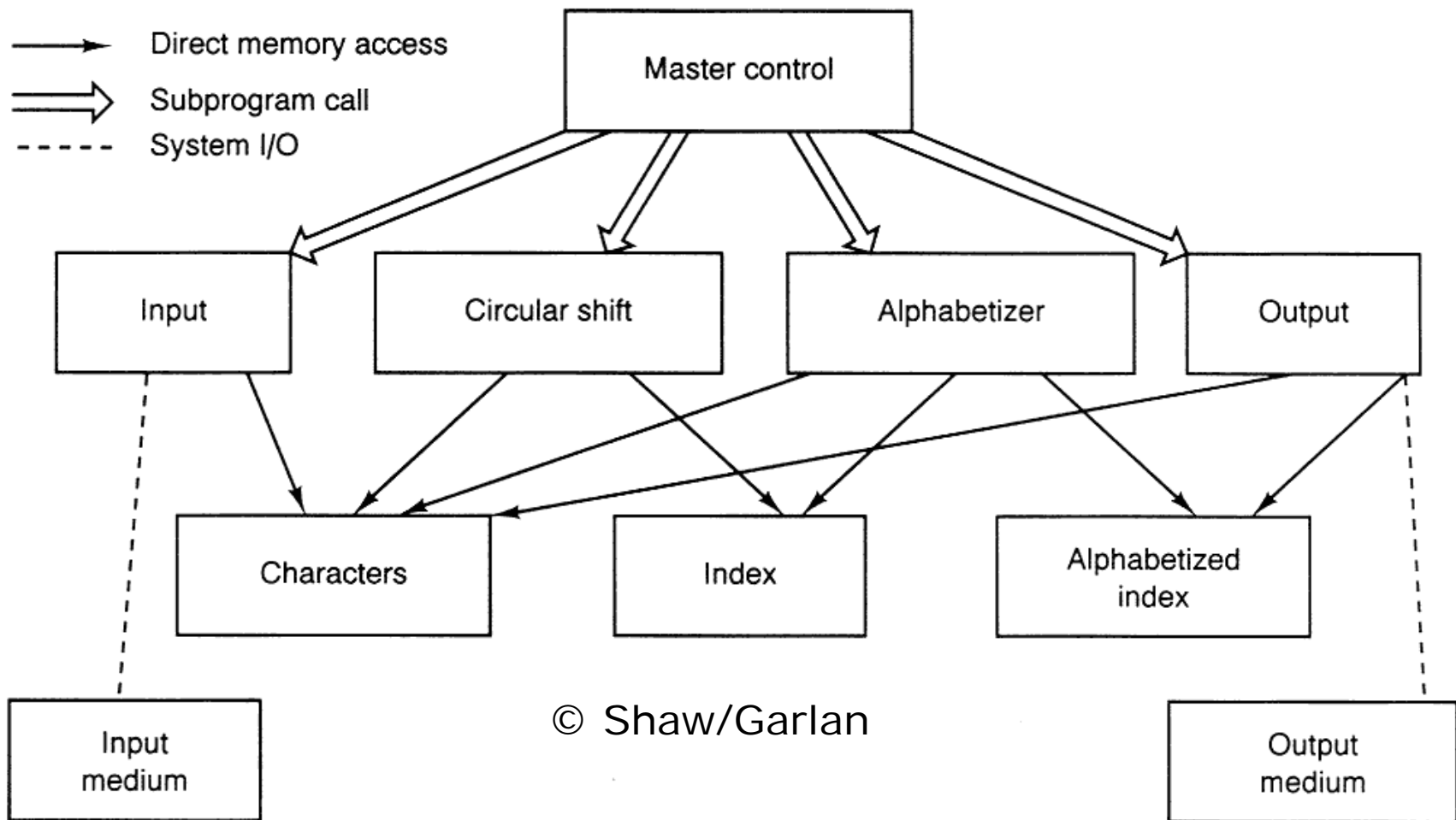


- Trotz der geringen Größe von KWIC stellen die drei Versionen Vertreter verschiedener Architekturen dar:
- KWIC 1 ist eine Datenflussstruktur (*pipe-and-filter*)
  - lediglich ein wenig objektbasiert verbrämt

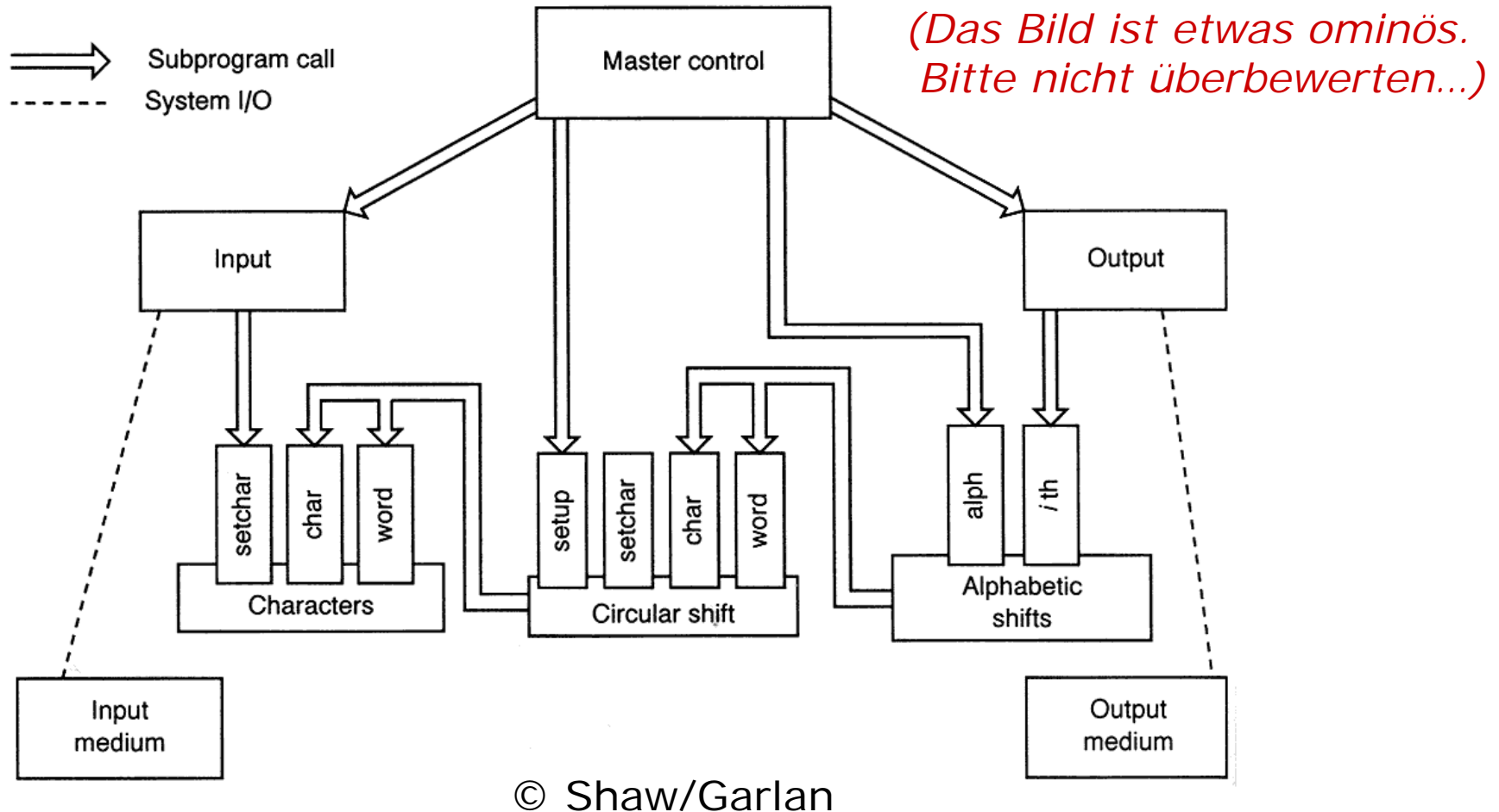


© Shaw/Garlan

- KWIC 2 ähnelt z.T. einer ablagebasierten Lösung (gemeinsame Nutzung von *TreeSet* durch *Rotator* u. *Writer*)
  - auch wieder durch Objektbasierung halbwegs verschleiert



- KWIC 3 ist eine Lösung mit Datenabstraktion



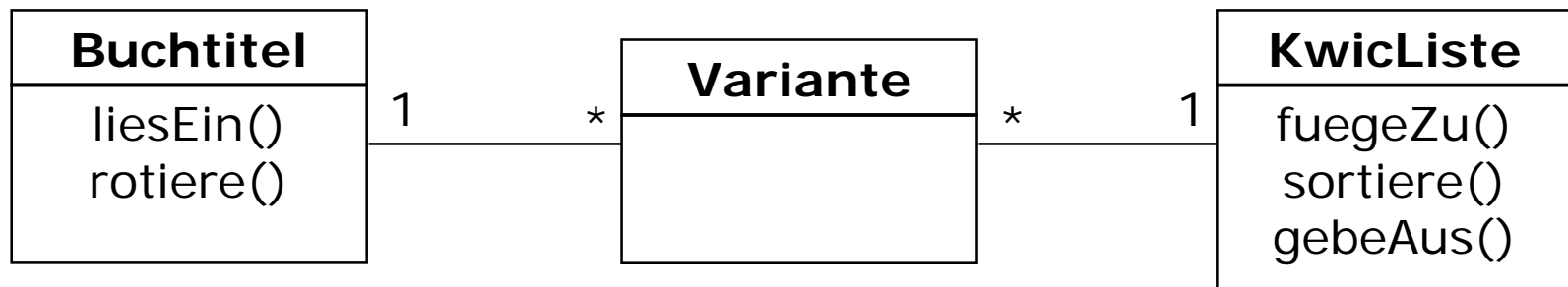
# Klassen(diagramme) versus Komponenten(diagramme)

- Bei der Anforderungsanalyse wird oft schon ein statisches Objektmodell angefertigt
- Verwechseln Sie dies nicht mit einem Entwurf!:
  - Manche der Klassen aus dem Analysemodell entsprechen späteren Komponenten
    - die aber dann oft aus viel mehr als einer Klasse bestehen
    - Bedenke: Unser Beispiel war winzig!
  - Manche der Klassen werden völlig anders implementiert
    - z.B. wg. Wiederverwdg. (*Sorter!*) oder wg. nichtfunktionaler Anford.
  - Manche der Klassen gehören zusammen in eine Komponente
    - weil sonst wichtige Entscheidungen nicht verborgen wären
- Merke:
  1. Bei Analyseklassen steht nicht ein Geheimnis im Mittelpunkt
  2. Komponenten sind sehr häufig größer als nur eine Klasse

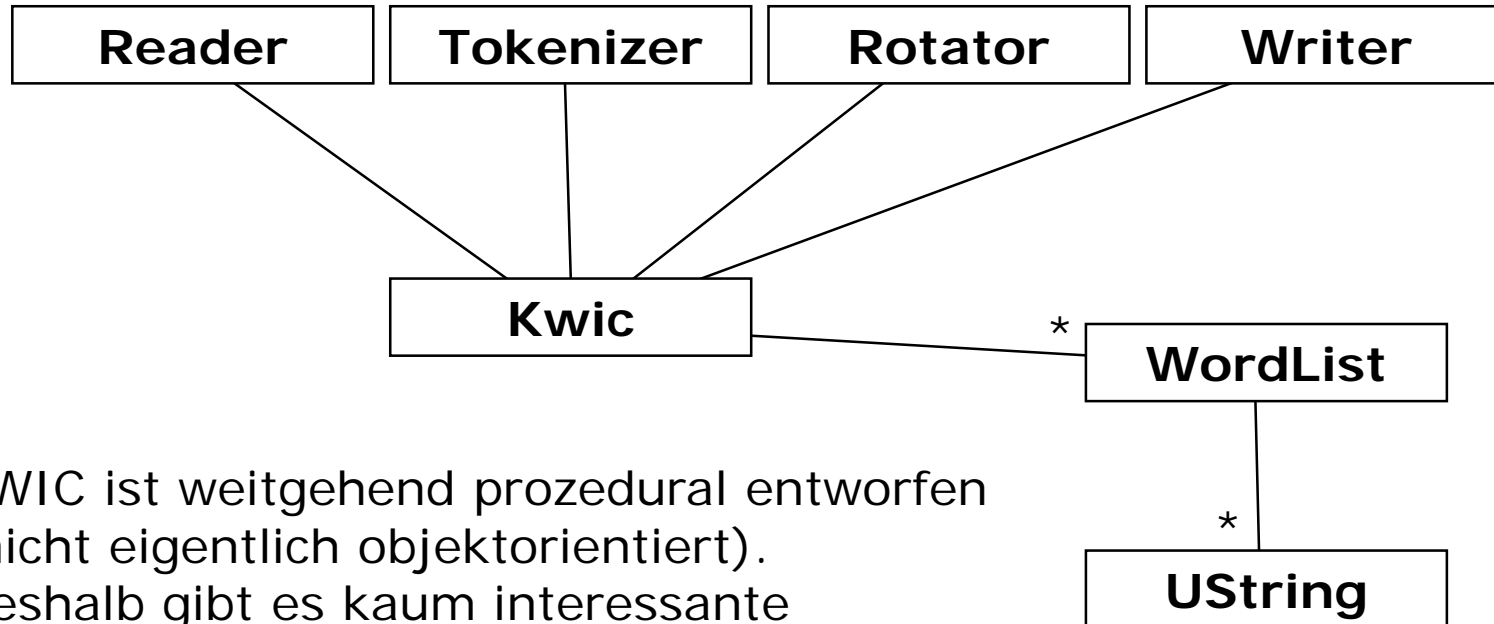
- 1. Analyse:
  - Identifiziere (einige) Klassen
  - Dies sind Problembereichsklassen (*application domain classes*), noch keine Lösungsbereichsklassen (*solution domain classes*)
- 2. Grobentwurf:
  - Wähle eine Architektur, identifiziere wichtige Entwurfsentscheidungen und bilde Komponenten
    - Jede Problembereichsklasse muss in irgendeiner Komponente aufgehen (es sei denn, sie entfällt ganz)
- 3. Feinentwurf:
  - Zerlege jede Komponente sukzessive in Teilkomponenten und schließlich in Klassen
  - Dies sind jetzt Lösungsbereichsklassen
  - Manche der Problembereichsklassen sind evtl. nicht mehr wieder zu erkennen (oder gar nicht mehr wieder zu finden)

# Klassen vs. Komponenten: KWIC

- Bedenken Sie z.B., dass alle drei Versionen von KWIC aus der selben Analyse hervorgehen!
- Das erste Analysemodell könnte z.B. so aussehen
  - das hängt aber sehr von der Formulierung der Anforderungen ab

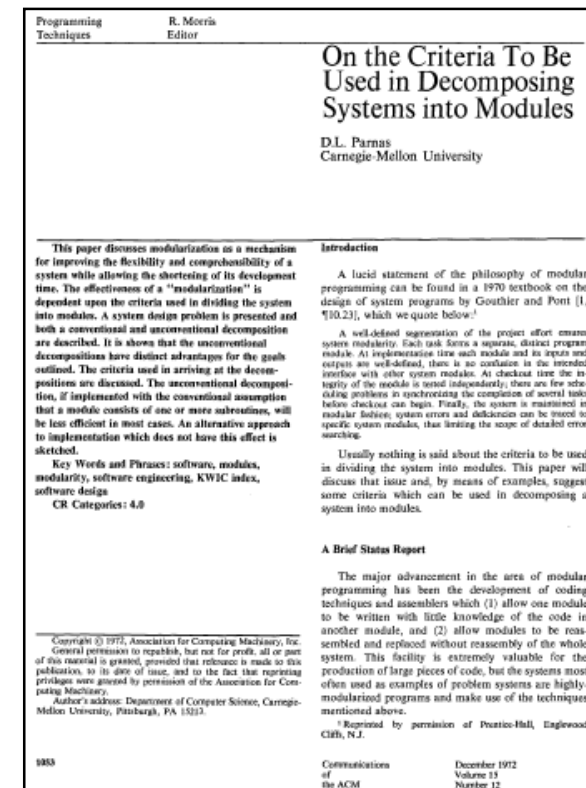


# KWIC 3 Klassendiagramm (Entwurf)



KWIC ist weitgehend prozedural entworfen (nicht eigentlich objektorientiert). Deshalb gibt es kaum interessante Beziehungen.

- Das KWIC-Beispiel ist angelehnt an einen klassischen Artikel:
  - David Parnas: **"On the criteria to be used in decomposing systems into modules"**, Communications of the ACM 15(12):1053-1058, December 1972
- In diesem Artikel wurde erstmals das Prinzip der Verbergung von Entwurfsentscheidungen ("information hiding") explizit formuliert.
  - Auch damals konnten viele Programmierer bereits das Konzept von Modulen, zerteilten ihr Programm aber recht willkürlich oder nach anderen Kriterien
  - Auch heute gilt noch: Viele Entwerfer verbergen nicht die wichtigen veränderlichen Entwurfsentscheidungen!
    - OO lenkt davon nämlich eher ab





1. Eine gute Modularisierung zu finden ist schwierig
  - Viele Aspekte spielen eine Rolle
  - Wenn man die entscheidenden Änderungen nicht vorhersehen kann, gelingt es allenfalls mit Glück
    - Deswegen sind Standard-Architekturen so nützlich:  
Da ist das Glück quasi fest eingebaut
2. Sinnvoll aussehende Modularisierungen sind oft nicht so günstig wie sie scheinen (siehe KWIC 1)
  - Siehe die Meinungsschwankungen in den Umfragen
3. Die Änderungsfreundlichkeit muss gegen die Entwurfskomplexität abgewogen werden
  - KWIC 2 ist garantiert besser als KWIC 1
  - Aber ob KWIC 3 einen Fortschritt darstellt, ist nicht klar, denn es ist aufwändiger als KWIC 2
4. Kisten-und-Pfeile-Bildchen reichen nicht aus, um einen Entwurf zu beschreiben oder zu verstehen

# Danke!

