

Software- Qualitätsmanagement

**Vorlesung im Modul 10-202-2319
Software-Management**

Sommersemester 2011

Prof. Dr. Hans-Gert Gräbe

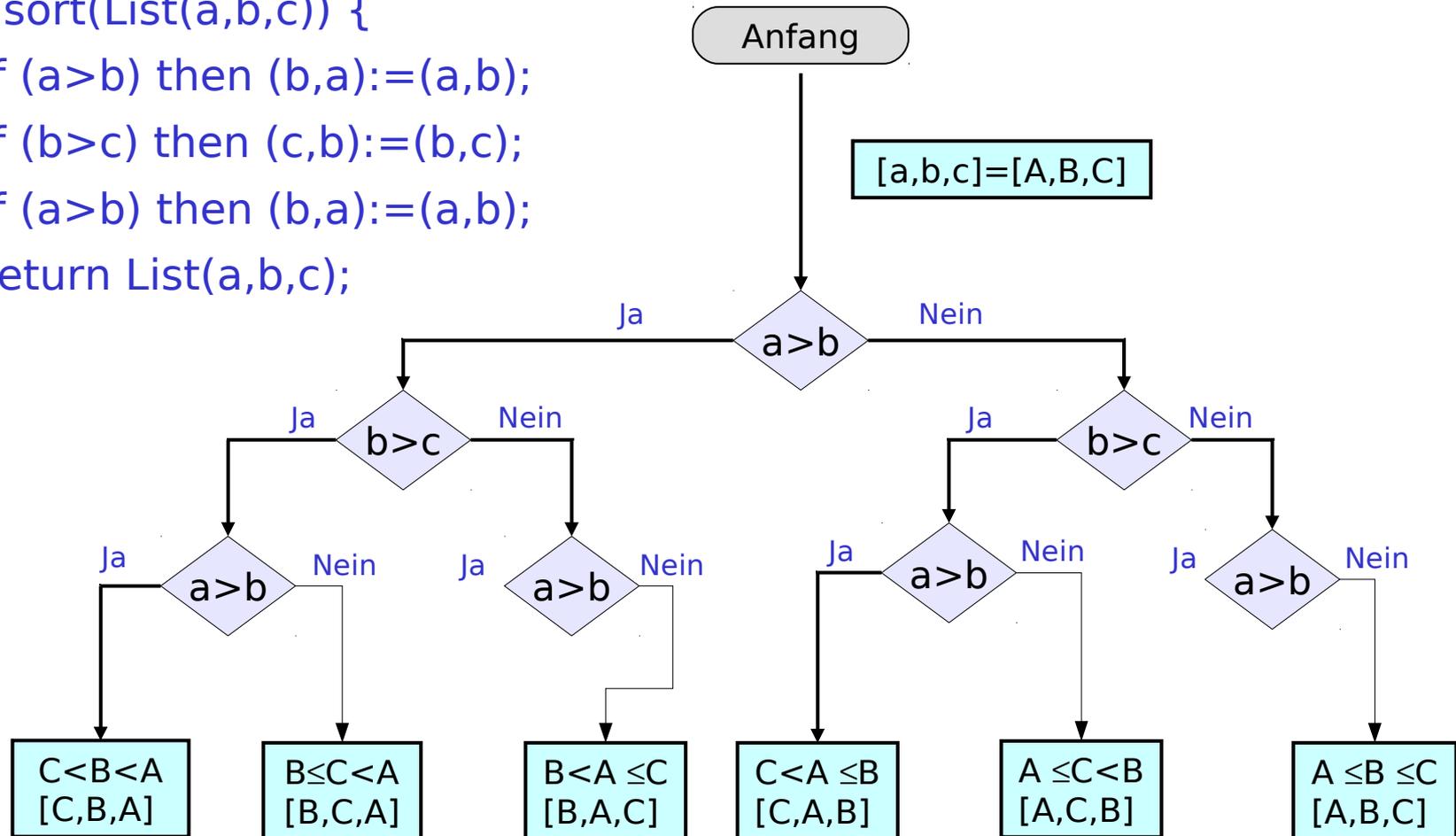
<http://bis.informatik.uni-leipzig.de/HansGertGraebe>

6. Verifizierende Verfahren

4. Symbolisches Testen

Beispiel

```
List sort(List(a,b,c)) {
  if (a>b) then (b,a):=(a,b);
  if (b>c) then (c,b):=(b,c);
  if (a>b) then (b,a):=(a,b);
  return List(a,b,c);
}
```



Quellcode-Analyse

Ansatz: Qualität von Systemkomponenten besteht nicht nur in deren **funktionaler Qualität** (Q.-Z. Funktionalität und Effizienz; Fokus der bisher besprochenen Qualitätssicherungs-Methoden Test und Verifikation), sondern auch in der **Qualität des Quellcodes** selbst (Q.-Z. Änderbarkeit, Übertragbarkeit sowie teilweise Benutzbarkeit).

Relevante Parameter:

- sinnvolle **Granularität** der Komponenten längs **funktionaler Grenzen**.
- sinnvolle **Schnittstellengestaltung** für die Zusammenarbeit der Komponenten untereinander.

Kann in quantitativen Parametern der **Bindung** (innerhalb einer Komponente) und **Kopplung** (zwischen Komponenten) erfasst werden.

Bindung und Kopplung

Die Bindung innerhalb einer Systemkomponente und die Kopplung der Systemkomponenten untereinander bestimmen die Struktur eines Software-Systems.

Bindung (*cohesion*) ist ein qualitatives Maß für die Kompaktheit einer Systemkomponente. Es werden dazu die Beziehungen zwischen den Elementen einer Systemkomponente betrachtet.

Kopplung (*coupling*) ist ein qualitatives Maß für die Schnittstellen zwischen den Systemkomponenten. Es werden der Kopplungsmechanismus, die Schnittstellenbreite und die Art der Kommunikation betrachtet.

- Je stärker die Bindungen der Systemkomponenten im Vergleich zu den Kopplungen, desto ausgeprägter ist die Struktur und Modularität eines Systems.
 - Bindung auf der Ebene der Funktionen (Methoden): Wie weit ist abgrenzbare Funktionalität an einer Stelle zusammengefasst?
 - Bindung auf der Ebene der Daten (Module und Klassen): Wie weit ist datenmäßig zusammengehörige Funktionalität zusammengefasst?
 - Informale Bindung (Pakete): Wie sind Datenabstraktionskonzepte an Datenstrukturen gebunden?
- Die Forderung nach guter Modularität wird erfüllt, wenn die Kopplungen minimiert und die Bindungen maximiert werden.
- Für **Komponenten** spielt der Bindungsgrad eine qualitätsrelevante Rolle, für **Systeme** die Ausgestaltung der Kopplung zwischen den Komponenten.

Bindung auf der Ebene der Funktionen und Methoden

- Gute Bindung liegt vor, wenn nur solche Elemente zu einer Einheit (Funktion bzw. Methode) zusammengefasst werden, die auch zusammen gehören.
- Bindung von Funktionen wird nur qualitativ erfasst.

Ziel: Erreichen einer guten funktionalen Bindung.

- Alle Elemente sind an der Verwirklichung einer einzigen, abgeschlossenen Funktion beteiligt.
- Komplexe Funktionen werden realisiert, indem Teilaufgaben an andere Funktionen delegiert werden, die selbst funktional gebunden sind (Manager-Funktionen).

Kennzeichen einer guten funktionalen Bindung:

- Alle Teile tragen dazu bei, ein einzelnes spezifisches Ziel zu erreichen.
- Es gibt keine überflüssigen Teile.
- Die Aufgabe kann mit genau einem Verb und genau einem Objekt beschrieben werden.
- Austausch gegen andere Funktion oder Methode, welche denselben Zweck erfüllt, leicht möglich.
- Hohe Kontextunabhängigkeit, d.h. einfache Beziehungen zur Umwelt.

Vorteile einer funktionalen Bindung:

- Hohe Kontextunabhängigkeit (die Bindungen befinden sich innerhalb der Prozedur, nicht zwischen Prozeduren).
 - Geringe Fehleranfälligkeit bei Änderungen,
 - Hoher Grad der Wiederverwendbarkeit,
 - Leichte Erweiterbarkeit und Wartbarkeit, da sich Änderungen auf isolierte, kleine Teile beschränken.
-
- Konzept der Bindung verallgemeinert die (konzeptuellen) Regeln für „guten Code“ zu Regeln für „guten Software-Entwurf“.
 - Die Bindungsart einer Prozedur lässt sich nicht automatisch ermitteln, sondern nur durch manuelle Prüfmethoden.
 - Diese Untersuchungen sind noch im experimentellen Stadium und haben heute weitgehend informellen (damit aber nicht weniger bindenden) Charakter.

Bindung auf der Ebene von Datenabstraktionen und Klassen

Beschreibt das Zusammenwirken verschiedener Funktionen, welche derselben Datenabstraktion oder Klasse zuzuordnen sind.

- Voraussetzung: Alle Methoden sind gut funktional gebunden

Gute Bindung einer Klasse (*model cohesion*) liegt vor, wenn

- sie ein einzelnes semantisch bedeutungsvolles Konzept repräsentiert,
- die Klasse keine verborgenen Klassen enthält und
- keine Operationen enthält, die an andere Klassen delegiert werden können.

Wird in der Literatur auch als Kohärenz bezeichnet.

Für Klassen ist weiter die Bindung innerhalb von Vererbungsstrukturen wesentlich.

Informale Bindung auf der Ebene von Paketen

Datenabstraktionen sollen dem Prinzip der guten informalen Bindung genügen.

- liegt vor, wenn mehrere, in sich abgeschlossene, funktional gebundene Zugriffsoperatoren, die zu einer Datenabstraktion gehören, auch nur auf einer einzigen Datenstruktur operieren.
- **Idee:** hinter der gemeinsamen Funktionalität liegt auch ein gemeinsames Datenmodell

Merkmale:

- Unterstützt das Geheimnisprinzip, d.h. die Datenstruktur gehört nur zu einer Datenabstraktion,
- Änderungen der Datenstruktur tangieren nur eine Datenabstraktion,
- Problem der Vermischung von Zugriffsoperationen, die alle auf derselben Datenstruktur operieren, wird vermieden.

Bindung in Vererbungsstrukturen

- Die ganze Vererbungshierarchie muss untersucht werden.
- **Starke Vererbungsbindung** liegt vor, wenn die Hierarchie eine Generalisierungs-/Spezialisierungshierarchie im Sinne der konzeptuellen Modellierung ist.
- **Schwache Vererbungsbindung** liegt vor, wenn die Hierarchie nur zum "*code sharing*" verwendet wird.
- Das **Ziel** jeder neu definierten Unterklasse muss sein, ein einzelnes semantisches Konzept auszudrücken.

Analyse der Kopplung

Typische Ansätze für Kopplungsmetriken

- **fan-in:** Gemessen wird die Anzahl der Komponenten, welche die Funktionalität einer zu vermessenden Komponente verwenden.
- **fan-out:** Gemessen wird die Anzahl der von einer Systemkomponente benutzten anderen Komponenten sowie die Anzahl der Datenstrukturen, welche durch die betrachtete Systemkomponente aktualisiert werden.

Wir kommen darauf später zurück

OO-Spezifik: Vererbung als Bindung oder Kopplung?

Vererbung als Kopplung:

- gute Vererbungsstruktur hat enge Kopplung, gute Systemstruktur möglichst lose Kopplung

Vererbung als Bindung:

- gute Systemstruktur hat enge Bindung

Vererbungsmetriken werden deshalb den Komponenten zugerechnet

Die experimentellen Erfahrungen legen folgende Zielgrößen für OO-Systeme nahe:

geringer fan-out-Wert

- Grund: Delegationprinzip sinnvoll einsetzen

hohe fan-in-Werte

- Grund: hohe Verwendbarkeit deutet auf gute Struktur hin
- geht nicht global, da $\text{Summe fan-in} = \text{Summe fan-out}$

relativ wenige Objekte sollten als Parameter übergeben werden

- Grund: Objekt kapselt Zustand, verhält sich also wie globale Variable

Einführung

Quantitative Aussagen über die Produktqualität einer Systemkomponente können mit Hilfe von **Metriken** ermittelt werden.

- Mit solchen Metriken sind heute nur einfache Aussagen über Eigenschaften einer Komponente möglich.
- Eine Metrik bewertet ein Software-System immer nur unter einem sehr speziellen Blickwinkel.
- Aussagekräftiger Gesamteindruck von einer Systemkomponente nur durch Auswertung einer Gruppe von Metriken, oft auch nur im Vergleich zu Parametern anderer, bereits im Einsatz befindlicher Komponenten.
- Metriken können nicht nur für bereits implementierte Komponenten, sondern auch schon entwicklungsbegleitend eingesetzt werden.

Metriken zum Erfassen der prozeduralen Komplexität

Ziel: Bewertung eines Produkts (Entwurfsdokuments, Grob/Fein-Entwurf, Code, Designdokumentation usw.) mittels Metriken

Schwerpunkt: Zuverlässigkeit, Änderbarkeit

Umfangsmetriken

- sind die ältesten Metriken
- stellen ab auf die textuelle Komplexität
- verwenden einfach verfügbare Informationen (Anzahl an Programmzeilen, Dateigröße, Zahl der Funktionen, ...)
- Vertreter: LOC, Halstead-Metrik, Function Points (zur Erfassung des Umfangs verbaler Anforderungen)

Logische Strukturmetriken (Kontrollfluss-Metriken)

- Analyse des Kontrollfluss-Graphen
- Wird samt seiner Begleitobjekte (Symboltabelle) sowieso vom Compiler ausgewertet
- Vertreter: McCabe-Metrik

Datenstrukturmetriken

- messen die Anzahl an Variablen, deren Gültigkeit und Lebensdauer sowie die Referenzierung der Variablen

Stilmetriken

- messen ob die Programme richtig eingerückt wurden und ob die Namenskonventionen eingehalten wurden

Interne Bindungsmetriken

- messen die syntaktische Bindung durch Prüfen des Codes jeder Komponente

7. Analysierende Verfahren

2. Metriken für Komponenten

Beispiele

Einsatzgebiet	Kriterium	Metrik
Komponenten- analyse	Umfang	lines of code
	innere Struktur	Kontrollfluss- komplexität
	Schnittstelle	# Methoden pro Klasse Schnittstellenbreite

Einsatzgebiet	Kriterium	Metrik
Systemanalyse	Umfang	lines of code
	Kopplung	# Aufrufe in/aus Komponenten
	OO-Strukturierung	OO-Metriken
Prozessanalyse	Aufwandsoptimierung	Zeiterfassung
	Dokumentenqualität	entdeckte Fehler pro Seite
	Prüfprozessqualität	# vorab gefundener Fehler / # in der Sitzung gefundener Fehler

Umfangsmetriken - Die Halstead-Metrik

Misst die textuelle Komplexität eines Programms, indem die Zahl der verwendeten Funktionen und der verwendeten Variablen ins Verhältnis gesetzt werden.

- Es wird jeweils die Gesamtzahl (Programmtext) und die Zahl verschiedener Objekte (Symboltabelle) bestimmt.
- η_1, N_1 = Zahl der (verschiedenen) Funktionen, Operatoren, Symbole oder Schlüsselwörter (z. B.: +, -, *, /, **while**, **if**, ...)
- η_2, N_2 = Zahl der (verschiedenen) Variablen, Operanden ...

Interpretation:

- $\eta = \eta_1 + \eta_2$: Größe des Vokabulars
- $N = N_1 + N_2$: Länge der Implementierung

Vorteile:

- einfach zu ermitteln,
- bei jeder Programmiersprache verwendbar und
- gute Eignung der Metriken für die zu messenden Größen

Nachteile:

- nur der Implementierungsaspekt betrachtet und
- Mehrdeutigkeiten im Messansatz, z. B. bei den Klassifikationsregeln für Operatoren und Operanden

abgeleitete Größen:

- $D = \eta_1 / 2 \cdot N_2 / \eta_2$,
Parameter für die Schwierigkeit, den Code zu verstehen
- Interpretation:
 $N_2 / \eta_2 =$ durchschnittliches Vorkommen jeder Variablen,
 $\eta_1 =$ Anzahl der verwendeten Funktionen

```
int ZaehleVokale(String s) {  
    int VokalAnzahl; char Zchn; int i;  
    for(i=0; i < s.length(); i++) {  
        Zchn=s[i];  
        if ((Zchn == 'A') || (Zchn == 'E') || (Zchn == 'I') ||  
            (Zchn == 'O') || (Zchn == 'U')) VokalAnzahl++;  
    }  
    return VokalAnzahl;  
}
```

ZaehleVokale	1	int	3	()	9	++	2
VokalAnzahl	3	String	1	{}	2	[]	1
Zchn	7	char	1	;	8	==	5
s	3	for	1	=	2		4
i	5	if	1	<	1	.	1
Konstanten (6)	6	return	1	length	1		

$\eta_1=17, N_1=44, \eta_2=11, N_2=25, D=19.32$

Kontrollfluss-Metriken - Die McCabe-Metrik

- Misst die strukturelle Komplexität eines Programms, indem eine Grapheninvariante, die **zyklomatische Zahl** $V(G)$ des Kontrollflussgraphen, bestimmt wird.
- $V(G) = e - n + 2p$ mit
 - e = Anzahl der Kanten des Graphen
 - n = Anzahl der Knoten
 - p = Anzahl der Zusammenhangskomponenten
- Kontrollflussgraph wird für jede Prozedur aufgestellt ($p=1$).
- Für solchen Graphen gilt

$$V(G) = b + 1$$

mit b = Anzahl der Bedingungen.

- Zyklomatische Zahl ist additiv auf Komponenten.
- Lineare Teilstücke können zusammengezogen werden.

Vorteile:

- einfach zu berechnen,
- grobes Maß für die Kontrollflusskomplexität: je größer, desto weiter weicht der Kontrollfluss vom linearen ($V(G)=1$) ab.

Nachteile:

- unterschiedliche Programmmerkmale werden stark vereinfacht
- Quellprogramm als zentrales Messobjekt überbetont
- Es wird nur das Programmgerüst, nicht aber die Komplexität einzelner und verschachtelter Anweisungen berücksichtigt

7. Analysierende Verfahren

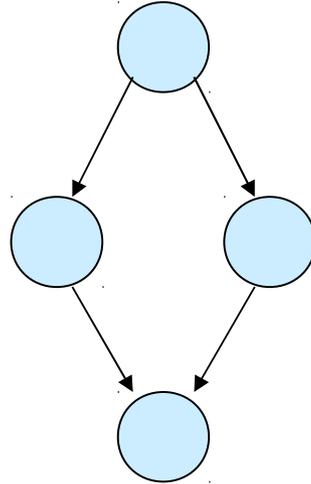
2.2 McCabe-Metrik - Beispiel

Sequenz



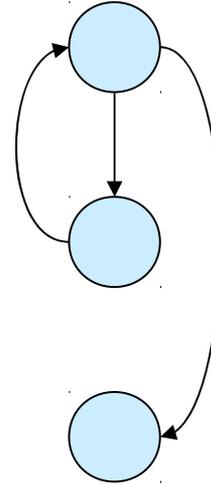
$$V(G) = 1 - 2 + 2 = 1$$

Auswahl



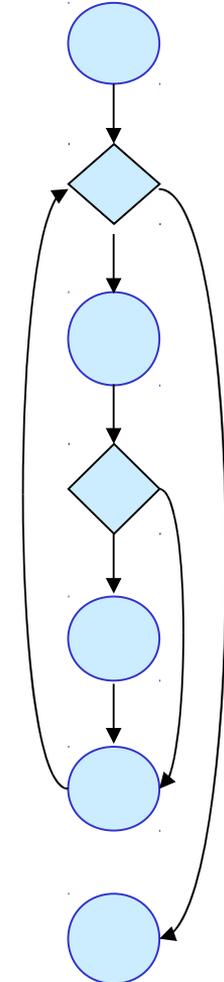
$$V(G) = 4 - 4 + 2 = 2$$

Abweisende Schleife



$$V(G) = 3 - 3 + 2 = 2$$

Das Programm ZaehleVokale hat die zyklomatische Zahl $V(G) = 8 - 7 + 2 = 3$
Es enthält zwei Bedingungen.



Metriken für objektorientierte Entwicklung

Metriken der klassischen Software-Entwicklung sind in unveränderter Form für OO-Projekte nur bedingt aussagefähig.

- Umfangsmetriken: Wie mit geerbtem Code umgehen?
 - Durch Vererbung und Polymorphismus sinkt die Zeilenzahl signifikant
- McCabe-Metrik: Kontrollflusskomplexität bei OO meist sehr gering

Frage nach Maßen für die OO-spezifischen Effekte

- zusätzliche Maße waren erforderlich
 - Breite und Höhe der Vererbungshierarchie
 - Anzahl der Klassen, die eine spezielle Operation erben
 - Anteil wieder verwendeter Komponenten
 - Anzahl der Objekt- und Klassenattribute
 - Anzahl der Objekt- und Klassenoperationen

Typische Metriken (Beispiele)

Objekt- und Klassenattribute einer Klasse:

- Anzahl: $|OV|$, $|CV|$
- gewichtete Anzahl: $\sum T(v)$
 - $T(v)$ = Gewicht des Typs des Attributs v
 - gewichtet nach Zahl der Vorkommen, nach Zahl der Vorkommen in verschiedenen Methoden der Klasse etc.

Objekt- und Klassenmethoden einer Klasse:

- $|OM|$, $|CM|$, evtl. wieder gewichtet nach Komplexität
- Parameterkomplexität einer Methode
 - Zahl und Gewicht der Typen der Aufrufparameter
- McCabe-Metrik

durchschnittliche Komplexität von Klassen in einem Paket

- Durchschnittswerte der einzelnen Klassenparameter
- Kreuzreferenzparameter (Bindungsanalyse im Paket)

Folgende Metriken werden in der Literatur als signifikant genannt

DIT (*Depth of Inheritance Tree*):

- Tiefe des Vererbungsbaumes (Zahl der Vorfahren einer Klasse)
- je höher der Wert von DIT, desto höher die Fehlerwahrscheinlichkeit (Hoher Nachnutzungsgrad, Nichtbeachtung verdeckter Annahmen)

NOC (*Number of Children of a Class*):

- Anzahl der direkten Nachfolger einer Klasse
- je höher der Wert von NOC, desto geringer die Fehlerwahrscheinlichkeit (sehr präzise Abstraktion, Nichtvorhandensein verdeckter Annahmen)

RFC (*Response For a Class*):

- Anzahl der Funktionen, die direkt durch die Methoden einer Klasse aufgerufen werden.
- je höher der Wert von RFC, desto größer die Fehlerwahrscheinlichkeit (Hoher Delegationsgrad, Nichtbeachtung verdeckter Annahmen)

WMC (*Weighted Methods per Class*):

- Anzahl aller neu definierten oder überschriebenen Methoden, die in jeder Klasse definiert sind.
- Geerbte Methoden werden nicht gezählt.
- je größer der Wert von WMC, desto höher die Fehlerwahrscheinlichkeit

CBO (*Coupling Between Object Classes*):

- eine Klasse ist mit einer anderen Klasse gekoppelt, wenn sie deren Methoden und/oder Attribute benutzt.
- CBO ist die Anzahl der Klassen, mit der eine Klasse gekoppelt ist.
- je größer der Wert von CBO, desto größer die Fehlerwahrscheinlichkeit (höherer Verschränkungsgrad)

Vorteile:

- erste Ansätze zur Verbesserung objektorientierter Komponenten
- erste empirische Untersuchungen zeigen die Eignung einiger Metriken als Qualitätsindikatoren

Nachteile:

- die Ziele der Metriken sind nur implizit erkennbar
 - Zielrelevanz der Metriken noch ungenügend untersucht
- keine Metriken für dynamische Aspekte
 - z. B.: Zustandsautomaten
- keine semantische Unterscheidung der Methoden
 - Standardoperationen (lesen, schreiben, erzeugen,...) sind weniger fehleranfällig als auszuprogrammierende "fachkonzeptspezifische" Methoden
- keine Berücksichtigung der Oberklassenqualität
 - ob eigene, geerbte oder fremde Methoden
 - ob Vererbung von Methoden aus Standardklassen oder eigenen
- keine Metriken, die eine "gute" Vererbungsstruktur prüfen