Software-Qualitätsmanagement

Vorlesung im Modul 10-202-2319 Software-Management

Sommersemester 2009

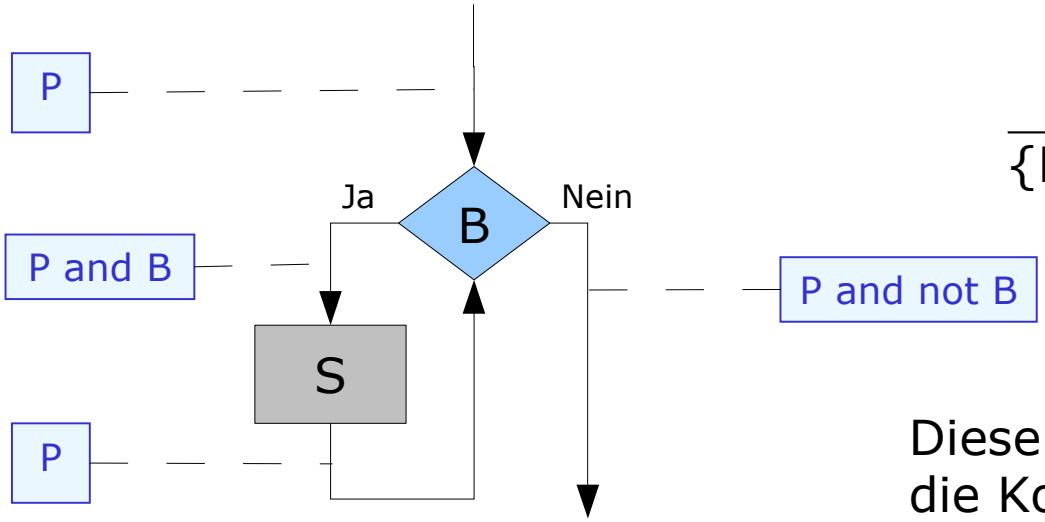
Prof. Dr. Hans-Gert Gräbe

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

while-Regel

- Bei der Verifikation von Schleifen spielt eine invariante Zusicherung **P**, die **Schleifeninvariante** eine entscheidende Rolle.
- Die Invariante gilt vor der Schleife und nach dem Schleifenrumpf.



$$\frac{\{P \text{ and } B\}\ S\ \{P\}}{\{P\}\ \text{while } B \text{ do } S\ \{P \text{ and not } B\}}$$

Diese Regel beweist nur **partiell** die Korrektheit der Schleife, denn die **Termination** wird durch P nicht garantiert.

Zum Beweis der Termination einer Schleife

- Wiederholungsbedingung B muss irgendwann falsch sein.
- Prüfung der Termination mit Hilfe einer **Terminationsfunktion t**.
  - **Idee**: Die Terminationsfunktion

    t : Programmzustände → **Z**

    ist nach unten beschränkt **und** wird in jedem Schleifendurchlauf kleiner.

Formale Formulierung der Bedingungen für t:

- { P and B and t = T } S { P and t < T } (T ist freie Variable)
- P and B ⇒ t ≥ 0

- **Variation**: Kettenbedingung auf Halbordnungen
  - Beispiel: Termordnungen auf dem Term-Monoid $T=T(x_1,...x_n)$
  - es reicht die Kettenbedingung statt Beschränktheit

Konditionierungsregel für Schleifen

Bei gegebener Invariante **P** und Terminationsfunktion **t** muss eine **while**-Schleife folgende Punkte erfüllen:

1. Die Invariante P muss während der Initialisierung der Schleife gesichert werden:
$$\{Q\} \text{ init } \{P\}$$
2. P bleibt im Schleifenrumpf S invariant, t wird bei jedem Ausführen des Schleifenrumpfes verringert.
$$\{P \underline{\text{and}} \ B \underline{\text{and}} \ t = T\} \ S \ \{P \underline{\text{and}} \ t < T\}$$
3. t ist vor jedem Ausführen des Schleifenrumpfes nicht negativ.
$$P \underline{\text{and}} \ B \Rightarrow t \geq 0$$
4. Die Nachbedingung R ist eine Folge der Schleifeninvariante.
$$P \underline{\text{and}} \ \underline{\text{not}} \ B \Rightarrow R$$

Entwickeln einer Schleife durch Weglassen einer Bedingung

Gegeben sei eine Spezifikation {Q} . {R: U and V}

1. R wird aufgeteilt in {R = P and not B}: P = U, B = not V
   - Die Invariante P ergibt sich durch Weglassen einer Bedingung.
   - Die weggelassene Bedingung {not V} wird zur Abbruchbedingung.
2. Initialisierung der Invarianten P durch ein Programmstück

   {Q} init {P}
3. Entwicklung eines Schleifenrumpfes mit der Spezifikation

   {P and B} S {P}
4. Hinzufügen der Terminationsbedingung **t**
   - **t** ergibt sich häufig aus dem Vergleich der Initialisierung mit der Abbruchbedingung {not B}.

Beispiel: *int y = isqrt(int x) mit y= ⌊sqrt(x)⌋*

$\{Q: x \geq 0\}$ und $\{R: y \geq 0 \text{ and } y^2 \leq x \text{ and } x < (y+1)^2\}$

Aufteilen von R: $\{P: y \geq 0 \text{ and } x \geq y^2\}$ und $\{B: x \geq (y+1)^2\}$

Initialisierung: $\{Q: x \geq 0\}\ y:=0\ \{P\}$

Schleife: $\{P \text{ and } B\}\ y:=y+1\ \{P\}$

$\{y \geq 0 \text{ and } x \geq (y+1)^2\}\ y`=y+1\ \{y` \geq 0 \text{ and } x \geq y`^2\}$

Terminationsfunktion: **t** := x − y

$\{P \text{ and } B \text{ and } t=T\}\ y:=y+1\ \{P \text{ and } t<T\}$

Java-Implementierung:

```
int isqrt(int x) {
    int y=0;
    while ((y+1)*(y+1) <= x)
        y=y+1;
    return y;
}
```

oder nach leichter Optimierung

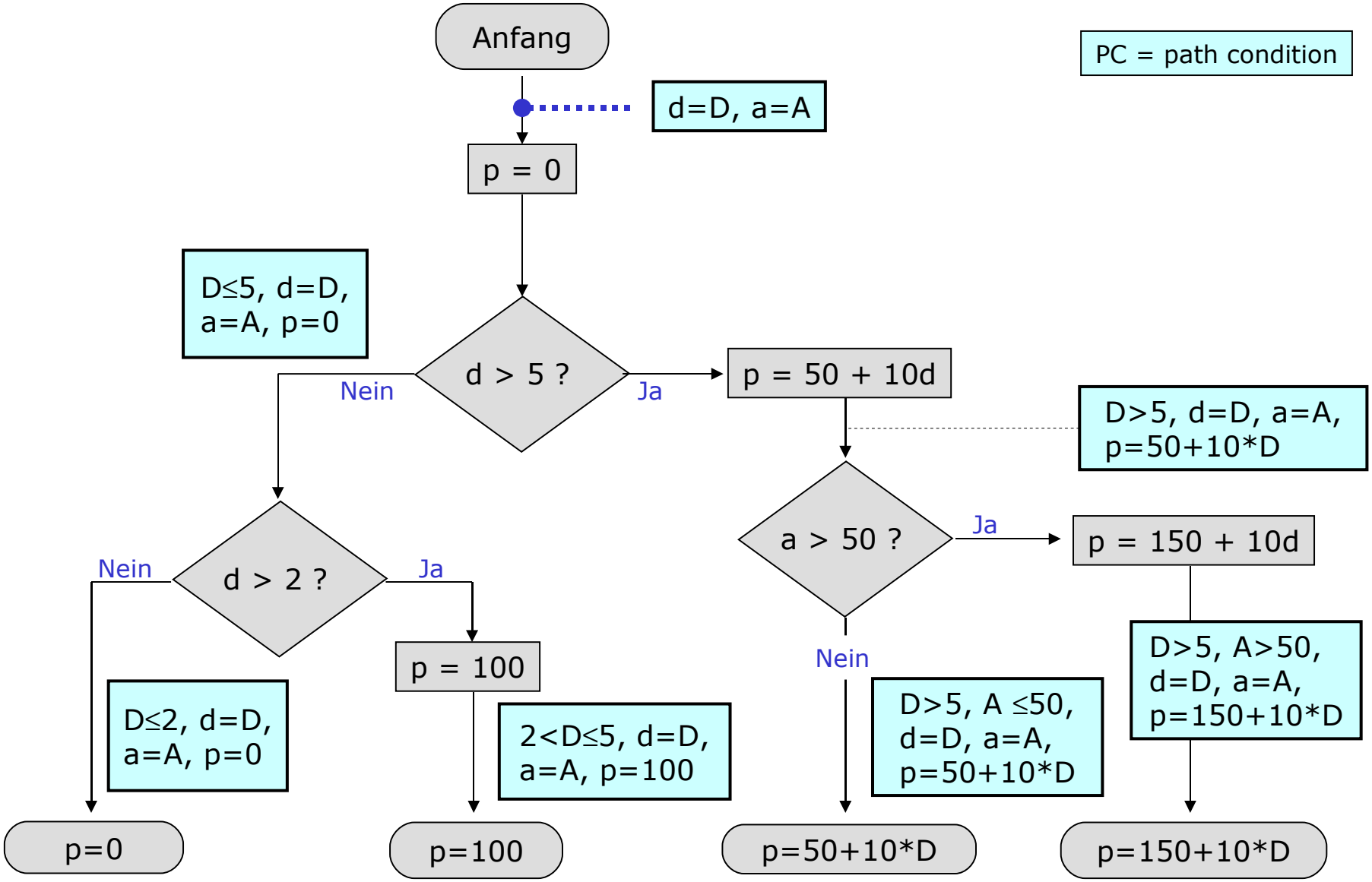
```
int isqrt(int x) {
    int y=1;
    while (y*y <= x)
        y=y+1;
    return (y-1);
}
```

Symbolisches Testen: Überblick

- **Idee**: Die Eingabeparameter des Programms werden mit symbolischen Variablen belegt und längs aller möglicher Kontrollflüsse alle möglichen **Zwischenergebnisse** und **Konditionen** in symbolischer Form bestimmt.
  - Methode ist besonders gut geeignet, wenn sich die an Verzweigungspunkten gültigen Kombinationen boolescher Bedingungen vereinfachen lassen und Zwischenergebnisse arithmetischer Natur sind.
- **Methode hat Beweiskraft** im mathematischen Sinn, wenn die symbolischen Parameter durch alle denkbaren konkreten Parameterwerte ersetzt werden können.
  - etwa darf das Zwischenergebnis y/x nicht ohne die Kondition x!=0 auftreten.
- **Schleifen** lassen sich in diesem Ansatz nur bedingt abbilden.

Beispiel

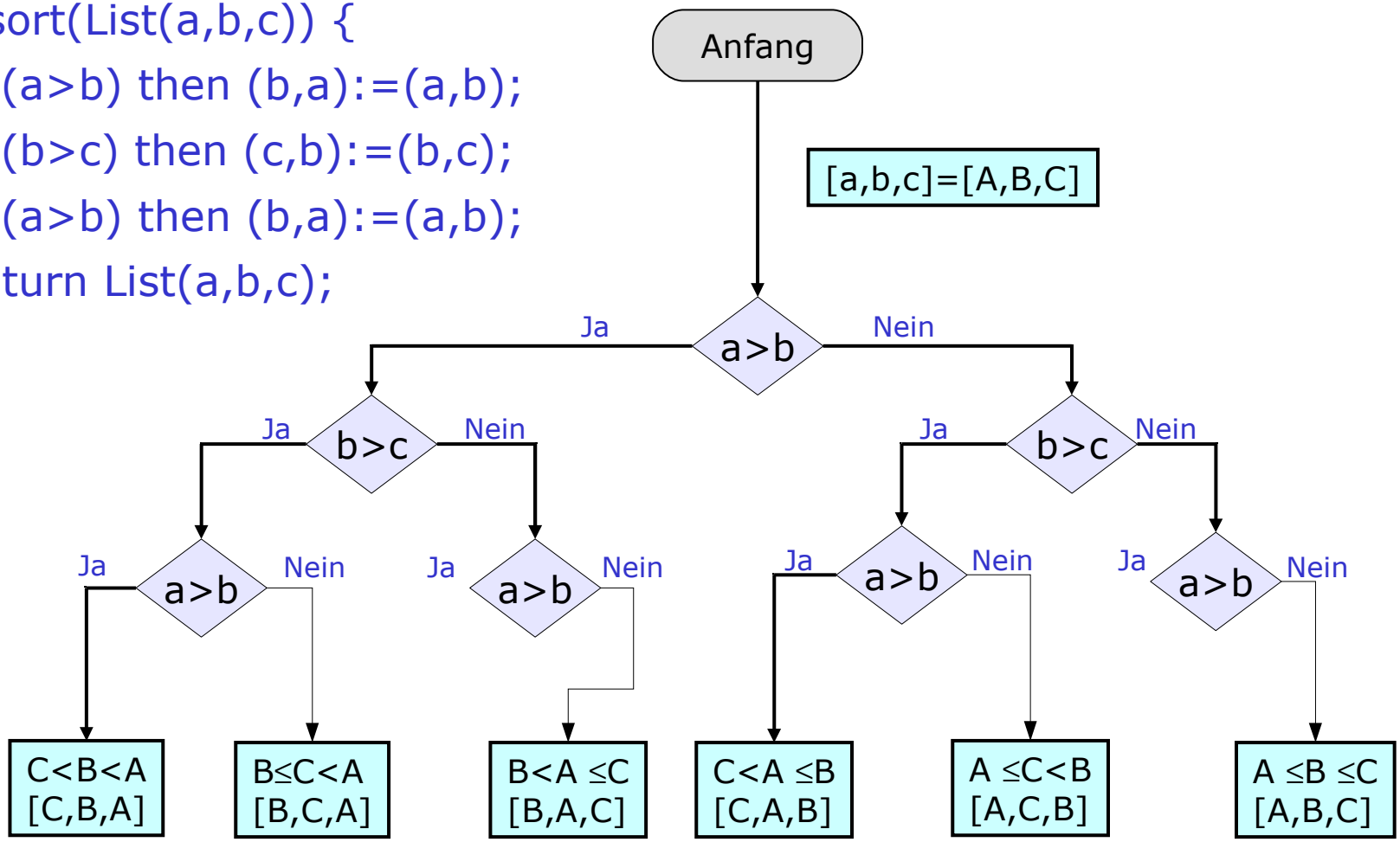
```
int berechnePraemie(int Dienstjahre, int Alter) {
    Praemie = 0;
    if (Dienstjahre > 5) {
        Praemie = 50 + 10 * Dienstjahre;
        if (Alter > 50) Praemie = Praemie + 100;
    }
    else if (Dienstjahre > 2) Praemie = 100;
    return Praemie;
}
```


Anfang
d=D, a=A
p = 0
D≤5, d=D, a=A, p=0
d > 5 ?
Nein
Ja
p = 50 + 10d
D>5, d=D, a=A, p=50+10*D
d > 2 ?
Nein
Ja
a > 50 ?
Ja
p = 150 + 10d
p = 100
Nein
D≤2, d=D, a=A, p=0
2<D≤5, d=D, a=A, p=100
D>5, A ≤50, d=D, a=A, p=50+10*D
D>5, A>50, d=D, a=A, p=150+10*D
p=0
p=100
p=50+10*D
p=150+10*D
PC = path condition

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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);
}
```



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Quellcode-Analyse

Ansatz: Qualität von Systemkomponenten besteht nicht nur in deren **funktioneller 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 **funktioneller Grenzen**.
- sinnvolle **Schnittstellengestaltung** für die Zusammenarbeit der Komponenten untereinander.

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- 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: Wie weit ist abgrenzbare Funktionalität an einer Stelle zusammengefasst?
  - Bindung auf der Ebene der Daten: Wie weit ist datenmäßig zusammengehörige Funktionalität zusammengefasst?
  - Informale Bindung: 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.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bindung auf der Ebene der Funktionen

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

Ziel: Erreichen einer funktionalen Bindung.

- Alle Elemente sind an der Verwirklichung einer einzigen, abgeschlossenen Funktion beteiligt.
- Komplexe Funktionen werden realisiert, indem importierte Funktionen verwendet werden, die selbst funktional gebunden sind.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Kennzeichen einer funktionalen Bindung:

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bindung von Datenabstraktionen/Klassen

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

- Voraussetzung: Alle Methoden sind 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.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Informale Bindung

Abstrakte Datenobjekte sollen dem Prinzip der informalen Bindung genügen.

- liegt vor, wenn mehrere, in sich abgeschlossene, funktional gebundene Zugriffsoperatoren, die zu einer Datenabstraktion gehören, 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, da alle auf derselben Datenstruktur operieren.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Metriken zum Erfassen der prozeduralen Komplexität

- **Umfangsmetriken**
  - sind die ältesten Metriken
  - stellen ab auf die textuelle Komplexität
  - verwenden einfach verfügbare Informationen (Anzahl an Programmzeilen, Dateigröße, Zahl der Funktionen, ...)
  - Vertreter: 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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- **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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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.
  - $\boldsymbol{\eta_1}$, $\mathbf{N_1}$ = Zahl der (verschiedenen) Funktionen, Operatoren, Symbole oder Schlüsselwörter (z. B.: +, -, *, /, **while**, **if**, …)
  - $\boldsymbol{\eta_2}$, $\mathbf{N_2}$ = Zahl der (verschiedenen) Variablen, Operanden ...
- **Interpretation**:
  - $\boldsymbol{\eta}=\eta_1+\eta_2$: Größe des Vokabulars
  - $\mathbf{N}=N_1+N_2$: Länge der Implementierung

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- **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/η_2 = durchschnittliches Vorkommen jeder Variablen,
    η_1 = Anzahl der verwendeten Funktionen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

```
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
VokalAnzahl	3
Zchn	7
s	3
i	5
Konstanten (6)	6

int	3
String	1
char	1
for	1
if	1
return	1

()	9
{}	2
;	8
=	2
<	1
length	1

++	2
[]	1
==	5
||	4
.	1

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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

$$\mathbf{V(G) = b + 1}$$

mit **b** = Anzahl der Bedingungen.

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

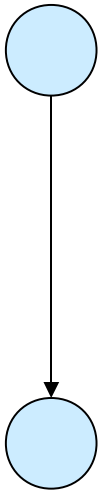
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- **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

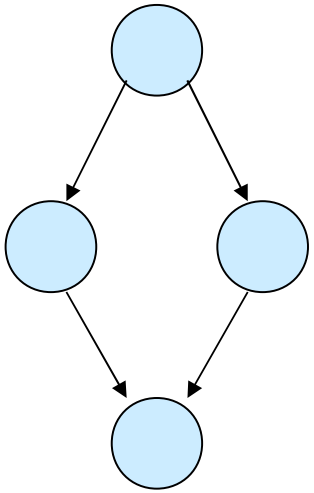
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Sequenz



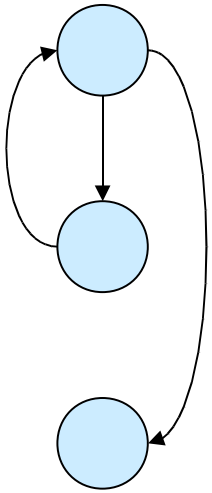
$V(G) = 1 - 2 + 2 = 1$

Auswahl

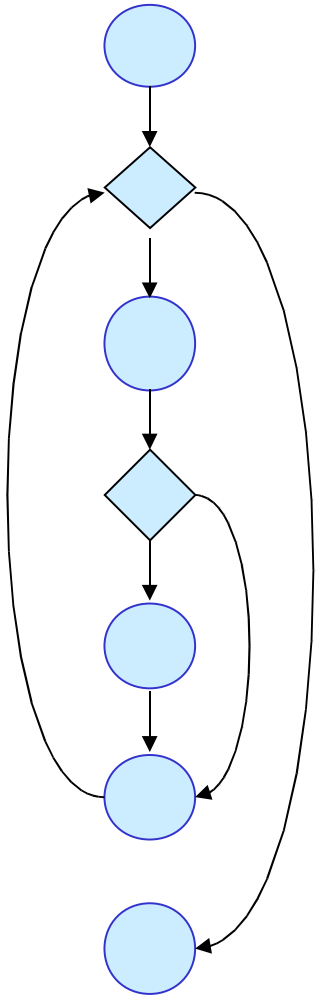


$V(G) = 4 - 4 + 2 = 2$

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.