Software-Qualitätsmanagement

Vorlesung im Modul 10-202-2319 Software-Management

Sommersemester 2012

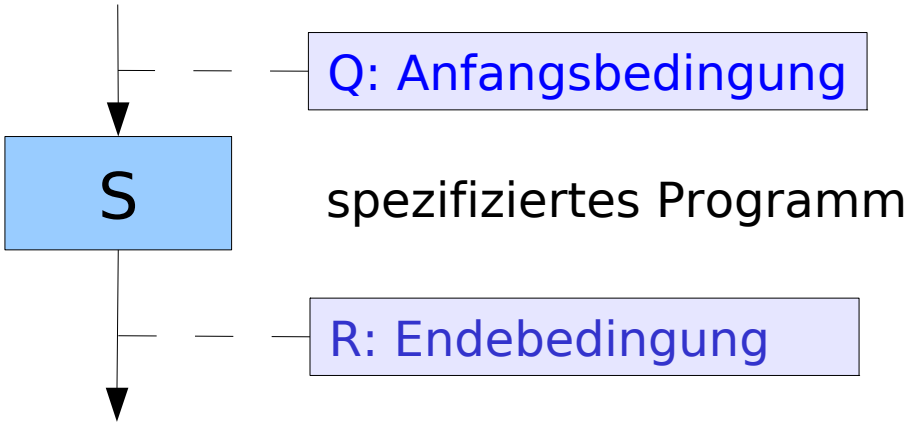
Prof. Dr. Hans-Gert Gräbe

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Konditionierung = Programm(teil) in einen Wenn-Dann-Zusammenhang einspannen

- Anfangsbedingung (Vorbedingung, *precondition*),
  - legt zulässige Werte der Variablen vor dem Ablauf des Programms fest
- Endebedingung (Nachbedingung, *postcondition*),
  - legt die gewünschten Werte der Variablen, sowie Beziehungen zwischen den Variablen nach dem Programmlauf fest.



Verifikation des Programms **{ Q } S { R }**
= Mathematisch exakter Beweis der Aussage

„Wenn vorher **Q** erfüllt ist und **S** ausgeführt wird, dann ist danach **R** erfüllt.“

Unterscheide

- zwischen **Zuweisung** und **Zusicherung**
  - Exponent := A (Zuweisung im Programmtext)
  - Exponent = A (Zusicherung im Kontext)
- zwischen **Programmvariablen** und **symbolischen Wertbezeichnern** sowie
- zwischen **Wert vor und nach der Zuweisung.**

Verifikationsregeln

- **Voraussetzung**: Programm ist aus konditionierten Bausteinen modular zusammengesetzt
  - Korrektheit des gesamten Programms ergibt sich aus der korrekten Zusammensetzung korrekter Teilstrukturen
- **Vorgehen**: Komplexes Programm wird verifiziert durch schrittweises Zusammensetzen aus **verifizierten einfacheren Strukturen** nach wenigen einfachen **Verifikationsregeln**.
- Folgende Verifikationsregeln existieren:
  - Konsequenz-Regel,
  - Zuweisungs-Regel,
  - Sequenz-Regel,
  - **if**-Regel und
  - **while**-Regel

Konsequenz-Regel

Gilt **{Q‘} S {R‘}** und

- Q‘ wird durch Q ersetzt, wobei Q schärfer ist als Q‘.
- R‘ wird durch R ersetzt, wobei R schwächer ist als R‘.

so gilt auch **{Q} S {R}.**

Beispiel:

$$\{Q' ::= x \leq y\}\ S\ \{R' ::= x = y+2\} \Rightarrow \{Q ::= x < y\}\ S\ \{R ::= y \leq x\}$$

- Geht man vorwärts durch ein Programm, so kann man Bedingungen abschwächen:
  - Hinzufügen eines Terms mit **oder**-Verknüpfung
  - Weglassen eines **und**-verknüpften Terms
    → Schwächere Bedingung
- Bei rückwärtiger Abarbeitung eines Programms, dürfen Bedingungen verschärft werden:
  - Hinzufügen eines Terms mit **und**-Verknüpfung
  - Weglassen eines **oder**-verknüpften Terms
    → Schärfere Bedingung
- Notation von Verifikationsregeln als Schlussregel:

$$\frac{\text{Voraussetzungen}}{\text{Schlussfolgerung}}$$

$$\frac{Q \Rightarrow Q',\ \{Q'\}\ S\ \{R'\},\ R' \Rightarrow R}{\{Q\}\ S\ \{R\}}$$

Zuweisungs-Regel

- Die Zuweisung x:=A verändert den Wert von x
  - Beispiel: { y+z = 25 } x:=y+z { x = 25 }
- Allgemeine Struktur: **{ R(A) } x:=A { R(x) }**
  - so zu verstehen: Hat man einen logischen Ausdruck R=R(x) mit der freien Variablen x und bildet Q::=R(A) durch Ersetzen dieser Variablen mit dem Ausdruck A, so ist die Aussage

    { Q } x:=A { R }

    wahr.
- Regel wird eingesetzt beim Rückwärtsarbeiten, um aus einer Nach- eine Vorbedingung abzuleiten:

  Bsp.: { Q? } x := x+25 { x = 2y }
  { R(A) } x‘ = x+25 { R(x‘) ::= x‘ = 2y}

  Als Vorbedingung ergibt sich { Q ::= 2y = x + 25 }

Sequenz-Regel

Zwei Programmteile S_1 und S_2 können zusammengesetzt werden, wenn die Nachbedingung von S_1 gleich der Vorbedingung von S_2 ist.

$$\frac{\{Q\}\, S_1\, \{P\}, \{P\}\, S_2\, \{R\}}{\{Q\}\, S_1 ; S_2\, \{R\}}$$

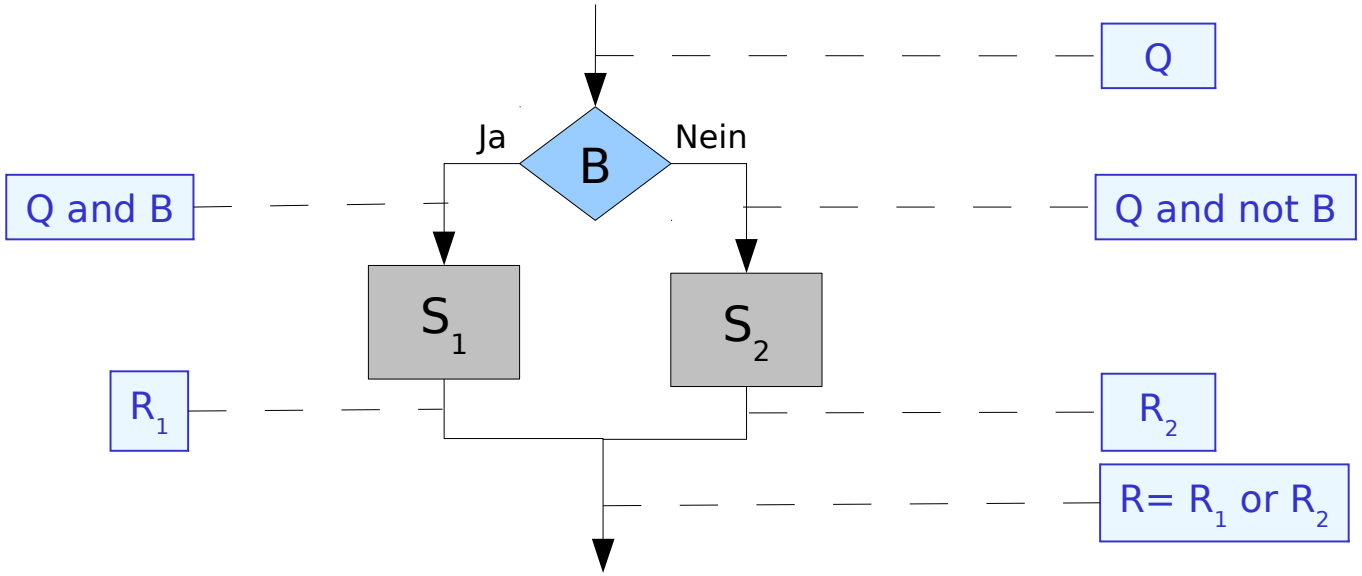
Die Sequenz-Regel kann mit Hilfe der Konsequenz-Regel noch verallgemeinert werden: Es genügt, wenn die Nachbedingung von S_1 „schärfer“ ist als die Vorbedingung von S_2, um S_1 und S_2 zu einem Programmstück zusammenzusetzen.

$$\frac{Q \Rightarrow Q_1, \{Q_1\}\, S_1\, \{R_1\}, R_1 \Rightarrow Q_2, \{Q_2\}\, S_2\, \{R_2\}, R_2 \Rightarrow R}{\{Q\}\, S_1 ; S_2\, \{R\}}$$

if-Regel

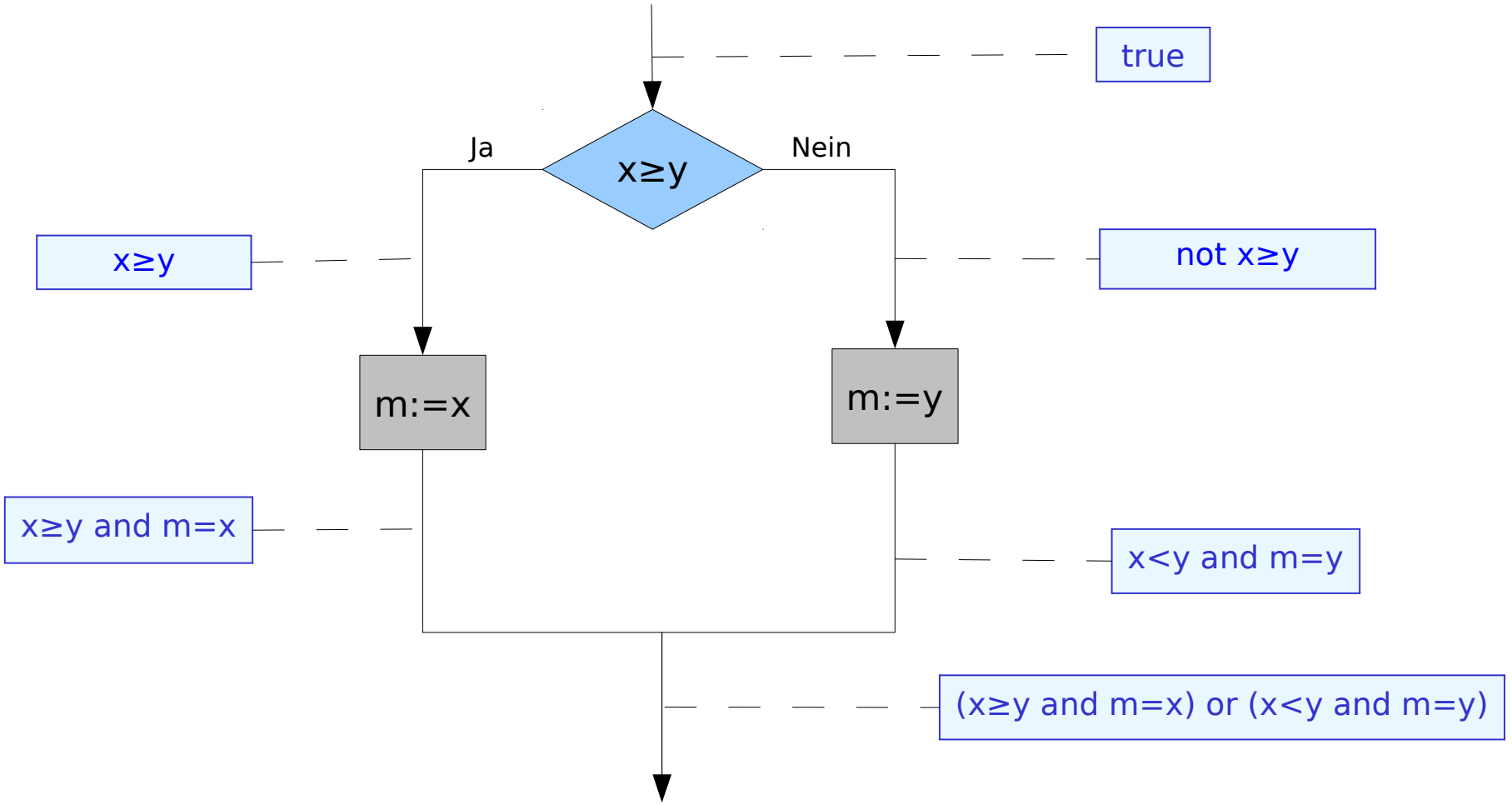
Gibt an, unter welchen Voraussetzungen zwei Programmstücke S_1 und S_2 und eine Bedingung B zu einer zweiseitigen Auswahl mit der Vorbedingung Q und der Nachbedingung R zusammengesetzt werden können.

$$\frac{\{Q \text{ and } B\}\ S_1\ \{R\},\ \{Q \text{ and not } B\}\ S_2\ \{R\}}{\{Q\} \text{ if } B \text{ then } S_1 \text{ else } S_2\ \{R\}}$$



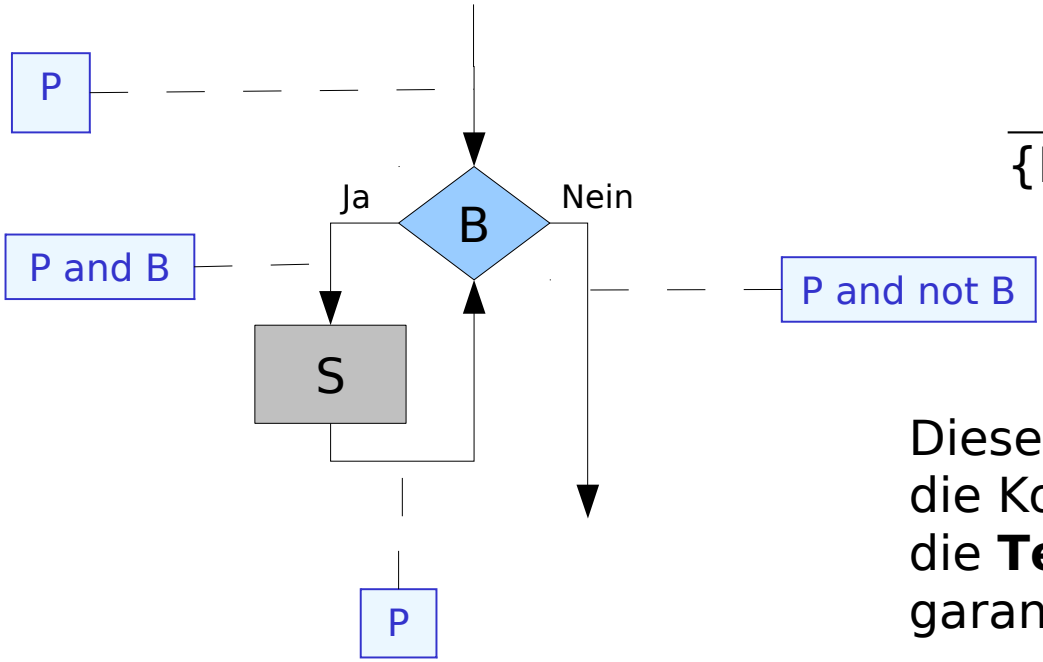
Beispiel:

{true} if x≥y then m:=x else m:=y {m=max(x,y)}



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

  $$\mathbf{t} : \text{Programmzustände} \rightarrow \mathbf{Z}$$

  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 $\Rightarrow$ t $\geq$ 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 ≥ 0} und {R: y ≥ 0 and y^2 ≤ x and x < (y+1)^2 }

Aufteilen von R: {P: y ≥ 0 and x ≥ y^2 } und {B: x ≥ (y+1)^2 }

Initialisierung: {Q: x ≥ 0} y:=0 {P}

Schleife: {P and B} y:=y+1 {P}

{y ≥ 0 and x ≥ (y+1)^2 } y‘=y+1 {y‘ ≥ 0 and x ≥ y'^2 }

Terminationsfunktion: **t** := x – y

{P and B and t=T} y:=y+1 {P 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);
}
```

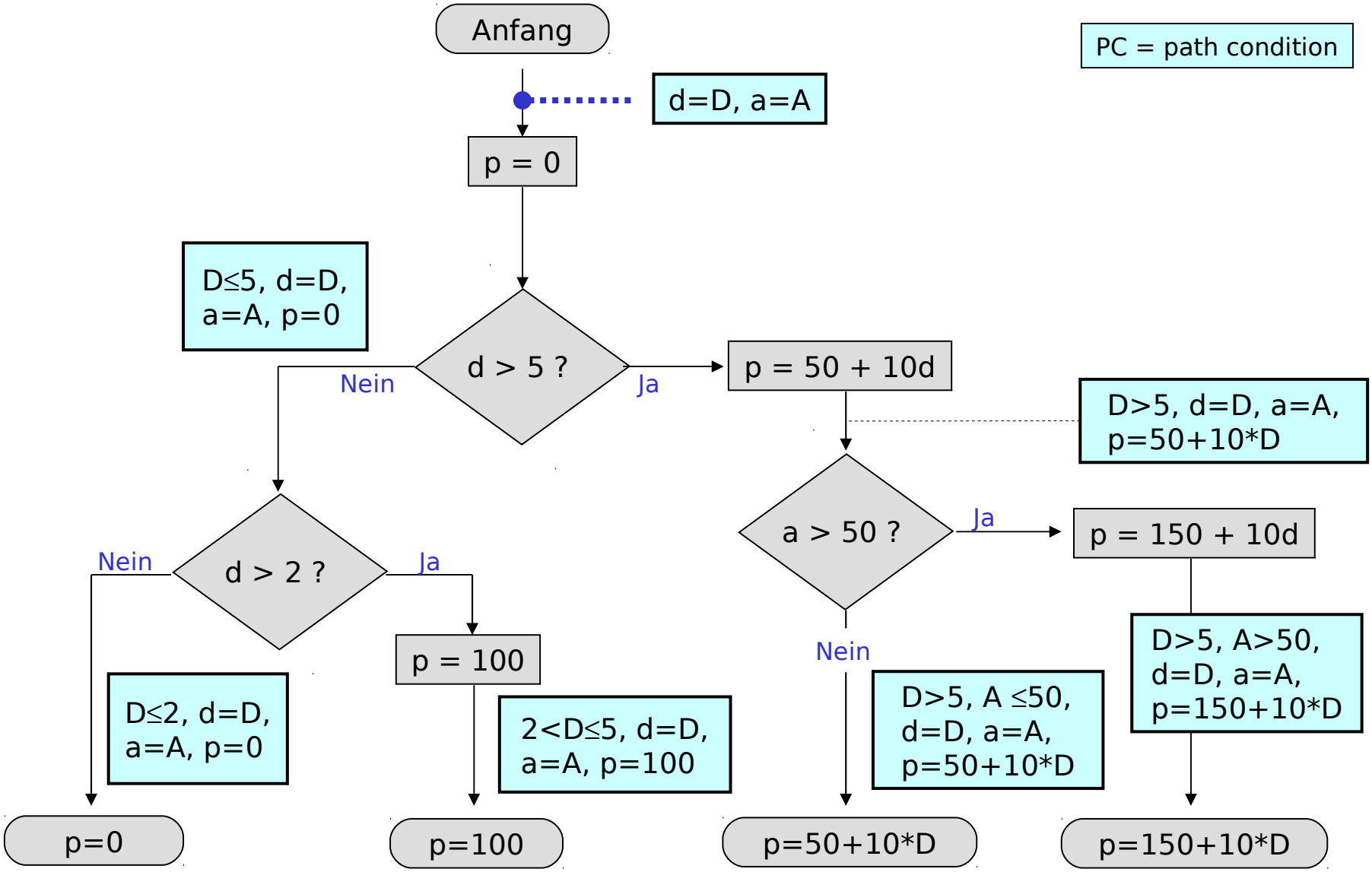
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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.

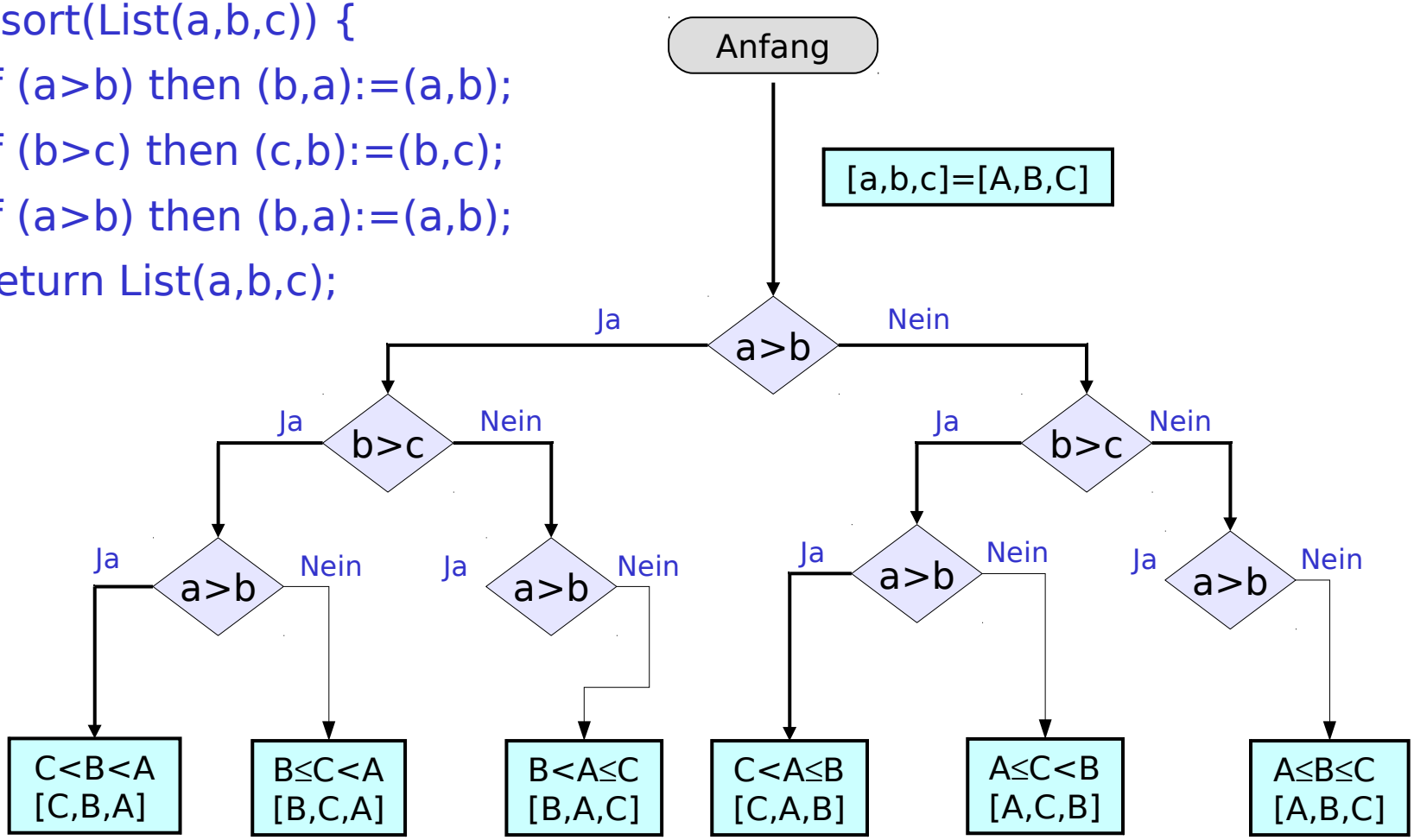
Beispiel

```
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

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 Kopplungs-mechanismus, 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 (Methoden): Wie weit ist abgrenzbare Funktionalität an einer Stelle zusammengefasst?
  - Bindung auf der Ebene der Datenabstraktionen (Module und Klassen): Wie weit ist datenmäßig zusammengehörende 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.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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).

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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.

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.
- Entsprechende Untersuchungen sind noch im experimentellen Stadium und haben weitgehend informellen Charakter.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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 (*model cohesion*) liegt vor, wenn

- die Klasse 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 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.

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

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.

Erfahrungen legen folgende Zielgrößen nahe:

geringer fan-out-Wert

- Grund: Delegierungsprinzip sinnvoll einsetzen

hohe fan-in-Werte

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

OO-Spezifik: Vererbung (extends) 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 Komponentenmetriken zugerechnet.

Implementations-Vererbung sollte nicht über Komponentengrenzen hinweg erfolgen, da die Korrumpierungsmöglichkeiten der Kopplungseffekte kaum zu überschauen sind.

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

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)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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

Beispiele

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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