

Vorlesung Software-Reengineering

Prof. Dr. Rainer Koschke

Arbeitsgruppe Softwaretechnik
Fachbereich Mathematik und Informatik
Universität Bremen

Wintersemester 2007/08

Überblick I

1 Metriken

- 1 Metriken
 - Softwaremetriken
 - Größenmetriken
 - Komplexitätsmetriken
 - Kopplung und Kohäsion
 - Objektorientierte Metriken
 - Empirische Studien
 - Wiederholungsfragen

- Kontext
 - mit Metriken können *Bad Smells* gefunden werden
 - mit Metriken kann man versuchen, Wartbarkeit zu quantifizieren
- Lernziele
 - verstehen, was eine Software-Metrik ist
 - klassische prozedurale und objektorientierte Software-Metriken kennen

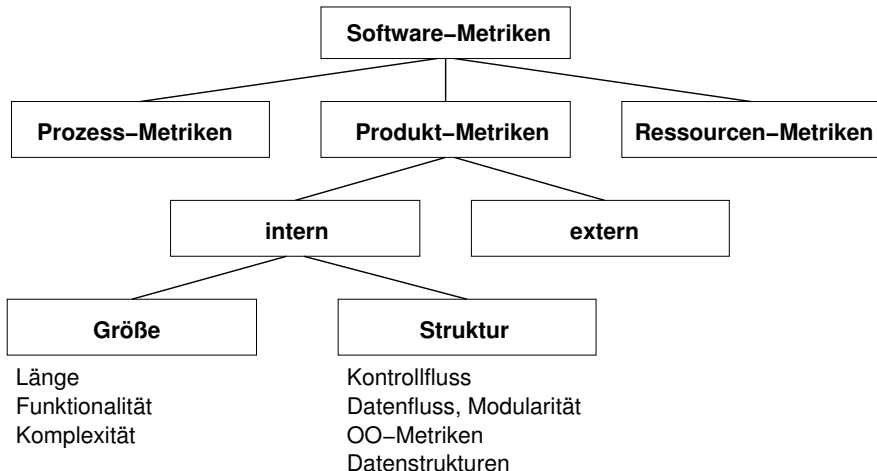
“A quantitative measure of the degree to which a system, component, or process possesses a given variable.”

– IEEE Standard Glossary

“A software metric is any type of measurement which relates to a software system, process or related documentation.”

– Ian Sommerville

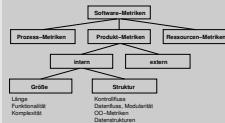
Klassifikation nach Fenton und Pfleger (1996)



Metriken

Softwaremetriken

Klassifikation nach Fenton und Pfleeger (1996)



Prozessmetriken - intern: Zeit, Aufwand, Anzahl gef. Fehler, ...

Prozessmetriken - extern: Qualität, Kosten, Stabilität, ...

Ressourcenmetriken - intern: Personal (Alter, Lohn), Teamgröße/-struktur, Hardwareausstattung, Büroeinrichtung,

...

Ressourcenmetriken - extern: Produktivität, Erfahrung, ...

Produktmetriken - extern: Verlässlichkeit, Verständlichkeit, Benutzerfreundlichkeit, Wartbarkeit, ...

Lines of code (LOC)

- + relativ einfach messbar
- + starke Korrelation mit anderen Maßen
- ignoriert Komplexität von Anweisungen und Strukturen
- schlecht vergleichbar

```
int main(int argc, char **argv) {  
    printf("Hello World."); }  
}
```

2007-12-03

Vorlesung Software-Reengineering

└─ Metriken

└─ Größenmetriken

└─ Größenmetriken – LOC

Größenmetriken – LOC

```
int main(int argc, char **argv) {  
    printf("Hallo World.");  
}
```

Wieviel LOC?

```
/*  
 * This program prints the message  
 * "Hello World." to stdout.  
 */  
  
int main(int    argc,  
          char **argv)  
{  
  
    printf("Hello World.");  
  
}
```

└─ Metriken

└─ Größenmetriken

└─ Größenmetriken – LOC

```
/*  
 * This program prints the message  
 * "Hello World." to stdout.  
 */  
  
int main(int argc,  
         char **argv)  
{  
  
    printf("Hello World.");  
  
}
```

Wie wird gezählt?

- Leerzeilen
- Kommentare
- Daten
- mehrere Anweisungen pro Zeile
- generierter Code
- lange Header usw.

⇒ Leerzeilen und Kommentarzeilen bei LOC nicht mitzählen, dafür Kommentarzeilen CLOC einzeln zählen; dann kann z.B. die Kommentardichte ermittelt werden als CLOC/LOC.

Nützlich zum Vergleichen von Projektgrößen, Produktivität, Entwicklung der Projektgröße, Zusammenhang mit Anzahl Fehler

Zählen per Modul, per Funktion, ...

```
/*  
 * This function should be documented.  
 *  
 * Author:  
 * Date created:  
 * Date modified:  
 * Version:  
 *  
 */
```

```
int main(int argc, char **argv)  
{  
    printf("Hello World.");  
}
```

Halstead (1977)

Länge	$N = N_1 + N_2$
Vokabular	$\mu = \mu_1 + \mu_2$
Volumen	$V = N \cdot \log_2 \mu$
Program Level	$L_{est} = (2/\mu_1) \cdot (\mu_2/N_2)$
Programmieraufwand	$E_{est} = V/L_{est}$

mit μ_1, μ_2 = Anzahl unterschiedlicher Operatoren, Operanden
 N_1, N_2 = Gesamtzahl verwendeter Operatoren, Operanden

- + komplexe Ausdrücke und viele Variablen berücksichtigt
- Ablaufstrukturen unberücksichtigt

Metriken

Größenmetriken

Größenmetriken – Halstead

Halstead (1977)

$$\text{Länge } N = N_1 + N_2$$

$$\text{Vokabular } \mu = \mu_1 + \mu_2$$

$$\text{Volumen } V = N \cdot \log_2 \mu$$

$$\text{Program Level } L_{\text{min}} = (2/\mu_1) \cdot (\mu_2/N_2)$$

$$\text{Programieraufwand } E_{\text{min}} = V/L_{\text{min}}$$

mit μ_1, μ_2 = Anzahl unterschiedlicher Operatoren, Operanden
 N_1, N_2 = Gesamtzahl verwendeter Operatoren, Operanden

- + komplexe Ausdrücke und viele Variablen berücksichtigt
- Ablaufstrukturen unberücksichtigt

Operanden = Variablen, Konstanten, Literale

Operatoren = Aktionen, bzw. alles andere außer Daten (+, *, while, for, ...)

Program Level = Größe der minimalen Implementierung / Größe der tatsächlichen Implementierung
 abgeleitete Halstead-Metriken umstritten.

Halstead: Zeitaufwand $T = E / 18$ Sekunden

```

int i, j, t;
if ( n < 2 ) return;
for ( i = 0; i < n-1; i ++ ) {
    for ( j = i + 1; j < n; j ++ ) {
        if ( a[i] > a[j] ) {
            t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }
}

```

<	=	>	-	,	;	()	[]	{	}	+	++	for	if	int	return
3	5	1	1	2	9	4	4	6	6	3	3	1	2	2	2	1	1

0	1	2	a	i	j	n	t
1	2	1	6	8	7	3	3

$$\mu_1 = 18, \mu_2 = 8, N_1 = 56, N_2 = 31$$

$$\Rightarrow V = N \cdot \log_2(\mu) = 87 \cdot \log_2(26)$$

```

int i, j, t;
if ( n < 2 ) return;
for ( i = 0; i < n-1; i++ ) {
  for ( j = i + 1; j < n; j++ ) {
    if ( a[i] > a[j] ) {
      t = a[i];
      a[i] = a[j];
      a[j] = t;
    }
  }
}

```

c	n	>	-	()	[]	{	}	**	for	if	ret.	return		
3	6	1	1	2	4	4	4	6	6	1	1	1	2	2	1	1

$\mu_1 = 18, \mu_2 = 8, N_1 = 56, N_2 = 31$
 $\Rightarrow V = N \cdot \log_2(\mu) = 87 \cdot \log_2(26)$

Wie werden Operanden + Operatoren definiert?

z.B. Operator = Token, Operand = Literal und Bezeichner

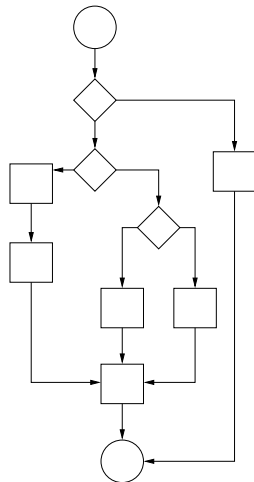
weitere:

- Anzahl Module
- Anzahl Operatoren, Operanden, Schlüsselworte
- Anzahl Parameter
- Anzahl/Umfang von Klonen
- durchschnittliche Länge von Bezeichnern
- ...

- Eigenschaften des Kontrollflussgraphen
- Eigenschaften des Aufrufgraphen (Größe, Tiefe, Breite)
- Modulkohäsion, Modulkopplung (Abhängigkeiten)
- OO-Metriken
- Daten, Datenstrukturen

Eigenschaften des Kontrollflussgraphen

- Anzahl Knoten
- Anzahl Kanten
- maximale Tiefe
- abgeleitete Maße



Komplexitätsmetriken – McCabe

Zyklomatische Komplexität (McCabe, 1976):
maximale Anzahl unabhängiger zyklischer Pfade
in stark verbundenen Graphen.

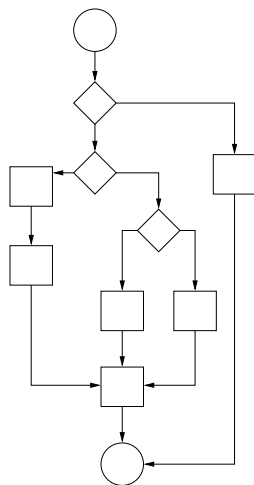
$$V(G) = \#Kanten - \#Knoten + 1^a$$

oder einfacher:

$$V(g) = \#Binärverzweigungen + 1$$

- + einfach zu berechnen
- Komplexität von Anweisungen unberücksichtigt

^aKontrollflussgraphen werden erst zu stark verbundenen Graphen durch eine künstliche Kante von Exit zu Entry $\rightarrow \#Kanten = \text{tatsächliche Kanten} + 1$



Metriken

Komplexitätsmetriken

Komplexitätsmetriken – McCabe

Zyklomatische Komplexität (McCabe, 1976):
maximale Anzahl unabhängiger zyklischer Pfade
in stark verbundenen Graphen.

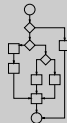
$$V(G) = \#Kanten - \#Knoten + 1^*$$

oder einfacher:

$$V(G) = \#Ein\ddot{a}r\ddot{a}r\ddot{z}weigungen + 1$$

- + einfach zu berechnen
- Komplexität von Anweisungen
unberücksichtigt

*Kontrollflussgraphen werden erst zu stark
verbundenen Graphen durch eine künstliche Kante von
Exit zu Entry → #Kanten := tatsächliche Kanten + 1



Zyklomatische Komplexität: maximale Anzahl unabhängiger zyklischer Pfade in stark verbundenen Graphen (strongly connected graphs).

Stark verbundener Graph: Jeder Knoten ist von jedem anderen Knoten erreichbar.

Wir nehmen an, jede Kante hat eine eindeutige Nummer. Jeder Pfad in einem Graph mit e Kanten kann durch ein e -Tupel (i_1, i_2, \dots, i_e) repräsentiert werden, bei dem der Index i_j angibt, wie oft die j -te Kante im Pfad vorkommt. Ein Pfad p ist eine Linearkombination von Pfaden p_1, \dots, p_n , wenn es ganze Zahlen a_1, \dots, a_n gibt, so dass $p = \sum a_j p_j$ ist, wobei die Pfade wie oben kodiert sind.

Eine Menge von Pfaden ist linear unabhängig, wenn kein Pfad eine lineare Kombination der anderen Pfade in der Menge ist.

Die Basismenge von Zyklen ist die maximal große Menge von linear unabhängigen Zyklen. Jeder Pfad eines zyklischen Graphen lässt sich als Linearkombination von Pfaden der Basismenge beschreiben.

Die Basismenge ist nicht notwendigerweise eindeutig. Allerdings ist die Kardinalität der Menge eindeutig. Sie wird zyklomatische Komplexität genannt und beträgt: $e - n + 1$.

Kontrollflussgraphen sind keine stark verbundenen Graphen, können jedoch leicht in einen solchen umgewandelt werden, indem der Exit-Knoten mit dem Entry-Knoten verbunden wird. Damit erhöht sich die Anzahl der Kanten um eins, so dass die zyklomatische Komplexität $e - n + 2$ beträgt.

Komplexitätsmetriken – McCabe

```
int i, j, t;
if ( n < 2 ) return;
for ( i = 0; i < n-1; i ++ ) {
    for ( j = i + 1; j < n; j ++ ) {
        if ( a[i] > a[j] ) {
            t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }
}
```

└─ Metriken

└─ Komplexitätsmetriken

└─ Komplexitätsmetriken – McCabe

```
int i, j, t;  
if ( n < 2 ) return;  
for ( i = 0; i < n-1; i++ ) {  
  for ( j = i + 1; j < n; j++ ) {  
    if ( a[i] > a[j] ) {  
      t = a[i];  
      a[i] = a[j];  
      a[j] = t;  
    }  
  }  
}
```

$$V(g) = 4 + 1 = 5$$

Empirische Untersuchungen über Zusammenhang zyklomatische Komplexität (ZK) und Wartungsaufwand:

- (Fenton und Ohlsson 2000): Korrelation von Fehlern und ZK vor Release (nicht jedoch nach Release)
- (Grady 1994): Korrelation von Änderungshäufigkeit und ZK; schlägt $ZK \leq 15$ als Qualitätsziel vor

Cyclomatic Complexity	Risk Evaluation
1-10	a simple program, without much risk
11-20	more complex, moderate risk
21-50	complex, high risk program
> 50	untestable program (very high risk)

http://www.sei.cmu.edu/str/descriptions/cyclomatic_body.html

Einsatz z.B. bei Qualitätssicherung für Software des Zuggunnels
England-Frankreich schreibt für Prozeduren $ZK \leq 20$ und $LOC \leq 50$ vor
(Bennett 1994).

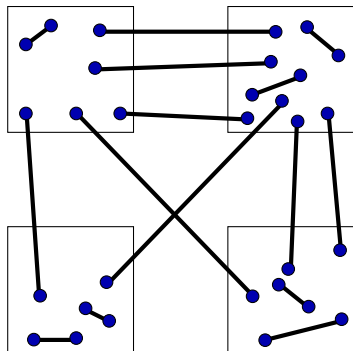
McCabe-Beispiele

```
case A is
  when 'A' => i := 1;
  when 'B' => i := 2;
  when 'C' => i := 3;
  when 'D' => i := 4;
  when 'E' => i := 5;
end case;
```

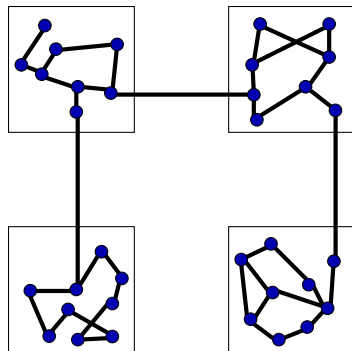
```
S : array (1..5) of Character := ('A', 'B', 'C', 'D', 'E');
i := 1;
loop
  exit when S(i) = A;
  i := i + 1;
end loop;
```

```
o = new ...;
...
o.mymethod (); // Verzweigung
```

Strukturmetriken – Kopplung und Kohäsion



starke Kopplung
schwache Kohäsion



schwache Kopplung
starke Kohäsion

OO-Metriken (Chidamber 1994; Chidamber und Kemerer 1994):

- WMC - weighted methods per class
- DIT - depth of inheritance tree
- NOC - number of children
- CBO - coupling between objects (uses, used-by)
- RFC - response for a class ($\#own + \#called$ methods)
- LCOM - lack of cohesion in methods

Metriken pro Klasse

Metriken

Objektorientierte Metriken

Strukturmetriken – OO

- OO-Metriken (Chidamber 1994; Chidamber und Kemerer 1994):
- WMC - weighted methods per class
 - DIT - depth of inheritance tree
 - NOC - number of children
 - CBO - coupling between objects (uses, used-by)
 - RFC - response for a class (#own + #called methods)
 - LCOM - lack of cohesion in methods

Metriken pro Klasse

WMC: Anzahl Klassenmethoden, optional gewichtet nach Größe oder Komplexität

DIT: Länge des Weges von der Wurzel bis zur Klasse (tiefe Hierarchie ist fehleranfällig)

NOC: Anzahl direkter Unterklassen, hohe Zahl ist Indikator für gute Wiederverwendung

CBO: Anzahl Klassen, mit denen eine Klasse gekoppelt ist (per uses, used-by); hoher Kopplungsgrad ist fehleranfällig, niedriger Kopplungsgrad fördert die Wiederverwendbarkeit

RFC: Anzahl Methoden, die potentiell ausgeführt werden können, wenn das Objekt auf eine eingehende Nachricht reagiert; RFC = #methods in the class + #remote methods directly called by methods in the class; bevorzugt: rekursiv

hoher RFC führt zu mehr Fehlern, deutet auf hohe Komplexität und schlechte Verständlichkeit hin

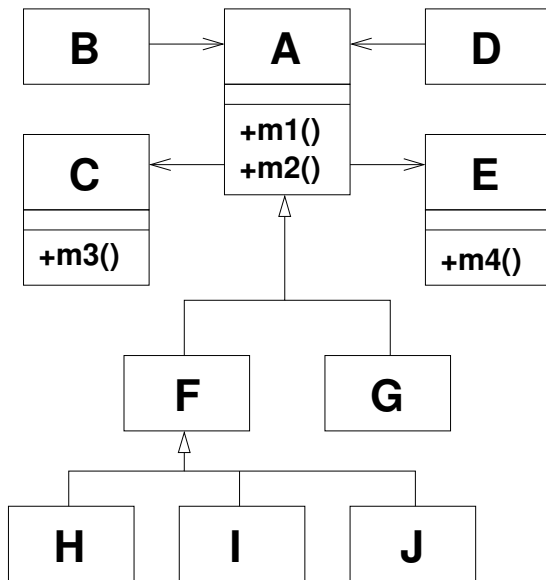
LCOM: hoher Wert heißt, Klasse führt mehrere Funktionen aus; deutet auf schlechtes Design, hohe Komplexität und hohe Fehlerwahrscheinlichkeit hin; Klasse sollte möglicherweise überarbeitet werden; niedriger Wert deutet auf gute Kapselung hin

LCOM1 = $\max(P-Q, 0)$, P=Anzahl der Paare von Methoden einer Klasse, die disjunkte Instanzvariablen benutzen; Q=Anzahl ... die mind. 1 Variable gemeinsam benutzen

LCOM2 = $1 - \frac{\sum(mA)}{m \cdot a}$ mit m=#Methoden, a=#Attribute, mA=#Methoden die ein Attribut ansprechen

LCOM3 = $(m - \sum(mA)/a) / (m-1)$

0 = hohe Kohäsion, 1 = keine Kohäsion, > 1 = Attribute werden nicht benutzt



2007-12-03

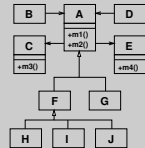
Vorlesung Software-Reengineering

└─ Metriken

└─ Objektorientierte Metriken

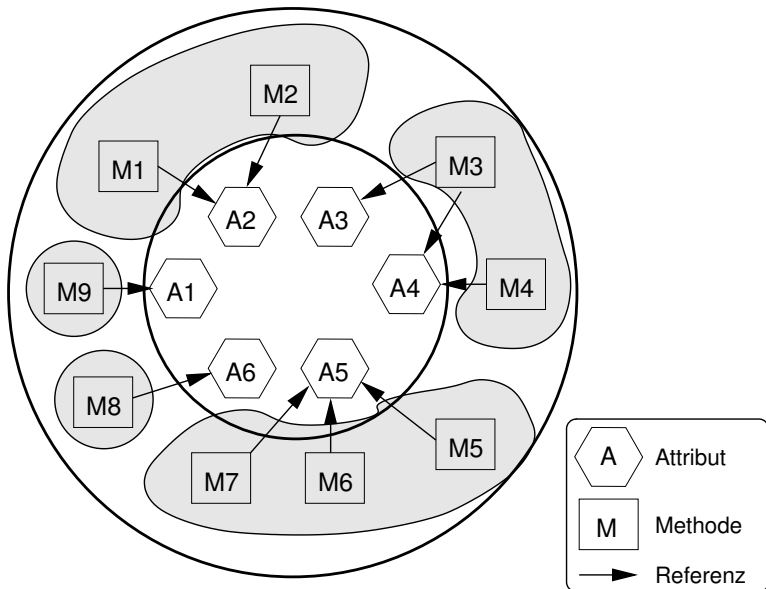
└─ Strukturmetriken – OO

Strukturmetriken – OO



$CBO(A) = 4$

$RFC(A) = 4, RFC(B) = 0, RFC(C) = 1$



Wie kann Qualität (Wartbarkeit, Testbarkeit, ...) gemessen werden?

- Bewertung der Qualität durch Entwickler
- Anwendung verschiedener Metriken
- Bestimmung der Korrelation
- Kombination der Metriken (orthogonal)

Maintainability Index (Coleman/Oman, 1994):

$$MI_1 = 171 - 5.2 \cdot \ln(V) - 0.23 \cdot V(g') - 16.2 \cdot \ln(LOC)$$
$$MI_2 = MI_1 + 50 \cdot \sin \sqrt{2.46 \cdot perCM}$$

- V = average Halstead Volume per module
- $V(g')$ = average extended cyclomatic complexity per module
- LOC = average LOC per module
- $perCM$ = average percent of lines of comment per module

- MI_2 nur bei sinnvoller Kommentierung
- $MI < 65 \Rightarrow$ schlechte / $MI \geq 85 \Rightarrow$ gute Wartbarkeit

- └─ Metriken

- └─┬─ Empirische Studien

- └─┬─┬─ Empirische Studien

Maintainability Index (Coleman/Oman, 1994):

$$MI_1 = 171 - 5.2 \cdot \ln(V) - 0.23 \cdot V(g^*) - 16.2 \cdot \ln(LOC)$$

$$MI_2 = MI_1 + 50 \cdot \sin \sqrt{2.46 \cdot \text{parCM}}$$

- ↳ V = average Halstead Volume per module
- ↳ $V(g^*)$ = average extended cyclomatic complexity per module
- ↳ LOC = average LOC per module
- ↳ parCM = average percent of lines of comment per module
- ↳ MI_2 nur bei sinnvoller Kommentierung
- ↳ $MI < 65 \Rightarrow$ schlechte / $MI \geq 85 \Rightarrow$ gute Wartbarkeit

nicht sinnvolle Kommentierung: wenn Kommentare nicht zum Code passen oder viel Code auskommentiert ist oder große Kommentarblöcke ohne relevante Informationen vorhanden sind.

Liso (2001): statt konstantem Faktor 2.46 abhängig von Programmiersprache wählen

Extended Cyclomatic Complexity: zusammengesetzte logische Ausdrücke werden berücksichtigt, AND und OR erhöhen die ECC jeweils um 1

Maintainability model (Muthanna u. a. 2000):

$$SMI = 125 - 3.989 \cdot FAN - 0.954 \cdot DF - 1.123 \cdot MC$$

- *FAN*: average number of external calls from the module
- *DF*: total number of outgoing and incoming data flow for the module
- *MC*: average McCabe for the module

Wartbarkeit korreliert mit

(Dagpinar und Jahnke 2003)

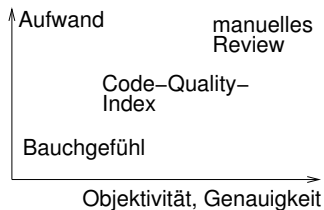
- TNOS - total number of statements
- NIM - number of instance methods
- FOUT - fan out, number of classes directly used

- **nicht** Vererbungshierarchie
- **nicht** Kohäsion
- **nicht** indirekte Kopplung
- **nicht** Kopplung über used-by Beziehungen

Testbarkeit korreliert vor allem mit (Bruntink und van Deursen 2004):

- LOCC - lines of code per class
- FOUT - fan out, number of classes directly used
- RFC - response for class

Bewertungsschema von Simon u. a. (2006)



- statisch ermittelter, objektiver Code-Quality-Index
- 52 Qualitätsindikatoren (Typen von Bad Smells)
- Auswirkungen der Qualitätsindikatoren auf Qualitätseigenschaften:
Analysierbarkeit, Modifizierbarkeit, Stabilität, Prüfbarkeit,
Austauschbarkeit, Zeitverhalten, Verbrauchsverhalten
- Häufigkeitsverteilung für mehr als 120 industrielle Systeme
geschrieben in C++ und Java → „Industriestandard“
- Quality-Benchmark-Level

Quality-Benchmark-Level

- ① Rudimentary: Code ist übersetz- und linkbar (nicht notwendigerweise ausführbar)
- ② Basic:
 - wirtschaftliche Anpassbarkeit gegeben
 - Analysierbarkeit und Stabilität besonders gewichtig
- ③ Extended:
 - gute technische Qualität
 - zusätzlich hohe Gewichtung von Zeitverhalten und Verbrauchsverhalten
- ④ Advanced:
 - hervorragende technische Qualität
 - zusätzlich hohe Gewichtung von Prüfbarkeit und Modifizierbarkeit
- ⑤ Complete:
 - perfekte technische Qualität
 - zusätzlich hohe Gewichtung von Austauschbarkeit

Bei jedem Ebenenwechsel: geringere Toleranz für Verstöße

- Hinweise auf Designfehler (Marinescu 2002; Lanza u. a. 2006)
- Klonerkennung (Mayrand u. a. 1996)
- Mustersuche (Reduzierung des Suchraums)
- ...

Freie Tools:

Tool	Sprachen	Metriken
clc (Perl)	C/C++	LOC, Comments, #Statements
cccc	C++, Java	LOC, McCabe, OO, ...
Metrics	C	LOC, Comments, Halstead, McCabe
MAS-C4	C	LOC, Comments, Halstead, McCabe, Nesting, Fan-In/-Out, Data Flow, ...
JMT	Java	OO-Metriken
Eclipse Plugins	Java	LOC, McCabe, OO, ...

Kommerzielle: z.B. Axivion Bauhaus Suite, McCabeQA, CMT, TAU/Logiscope, SDMetrics, CodeCheck, Krakatau Metrics, RSM, Together, ...

- Was ist eine Software-Metrik?
- Welche Produktmetriken kennen Sie (für die Größe, Komplexität und die Struktur)?

- 1 **Bennett 1994** BENNETT, P.A.: Software Development for the Channel Tunnel: a Summary. In: High Integrity Systems 1 (1994), Nr. 2, S. 213–220
- 2 **Bruntink und van Deursen 2004** BRUNTINK, M. ; DEURSEN, A. van: Predicting Class Testability Using Object-Oriented Metrics. In: Proceedings of the Fourth IEEE International Workshop on Source Code Analysis and Manipulation, IEEE Computer Society Press, September 2004
- 3 **Chidamber 1994** CHIDAMBER, S.: Metrics Suite For Object Oriented Design, M.I.T, Cambridge, Dissertation, 1994
- 4 **Chidamber und Kemerer 1994** CHIDAMBER, S.R. ; KEMERER, C.F.: A Metrics Suite for Object-Oriented Design. In: IEEE Computer Society Transactions on Software Engineering 20 (1994), S. 476–493
- 5 **Dagpinar und Jahnke 2003** DAGPINAR, M. ; JAHNKE, J.: Predicting Maintainability with Object-Oriented Metrics – An Empirical Comparison. In: Working Conference on Reverse Engineering, IEEE Computer Society Press, 2003

- 6 Fenton und Pfleger 1996** FENTON, N. ; PFLEEGER, S.: Software Metrics: A Rigorous and Practical Approach. 2nd. London : International Thomson Computer Press, 1996
- 7 Fenton und Ohlsson 2000** FENTON, Norman E. ; OHLSSON, Niclas: Quantitative Analysis of Faults and Failures in a Complex Software System. In: IEEE Computer Society Transactions on Software Engineering 26 (2000), August, Nr. 8, S. 797–814
- 8 Grady 1994** GRADY, R.B.: Successfully Applying Software Metrics. In: IEEE Computer 27 (1994), September, Nr. 9, S. 18–25
- 9 Lanza u. a. 2006** LANZA, Michele ; MARINESCU, Radu ; DUCASSE, Stephane: Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems. Berlin : Springer, August 2006. – ISBN ISBN-10: 3540244298, ISBN-13: 978-3540244295

- 0 Marinescu 2002** MARINESCU, Radu: Measurement and Quality in Object-Oriented Design, "Politehnica" University of Timisoara, PhD thesis, 2002. – URL
<http://loose.upt.ro/download/thesis/thesis.zip>
- 1 Mayrand u. a. 1996** MAYRAND, Jean ; LEBLANC, Claude ; MERLO, Ettore M.: Experiment on the Automatic Detection of Function Clones in a Software System using Metrics. In: Proceedings of the International Conference on Software Maintenance. Washington : IEEE Computer Society Press, November 4–8 1996, S. 244–254. – ISBN 0-8186-7678-7
- 2 Muthanna u. a. 2000** MUTHANNA, S. ; KONTOGIANNIS, K. ; PONNAMBALAM, K. ; STACEY, B.: A Maintainability Model for Industrial Software Systems Using Design Level Metrics. In: Working Conference on Reverse Engineering, IEEE Computer Society Press, 2000
- 3 Simon u. a. 2006** SIMON, Frank ; SENG, Olaf ; MOHNHAUPT, Thomas: Code-Quality-Management – Technische Qualität industrieller Softwaresysteme transparent und vergleichbar gemacht. dpunkt.verlag, 2006