Vorlesung Software-Reengineering

Prof. Dr. Rainer Koschke

Arbeitsgruppe Softwaretechnik
Fachbereich Mathematik und Informatik
Universität Bremen

Wintersemester 2010/11

Überblick I

Metriken

Software-Metriken I

Metriken

Fragen

- Wie lassen sich *Bad Smells* mit Metriken finden?
- Wie lassen sich Wartbarkeitsaspekte quantifizieren?

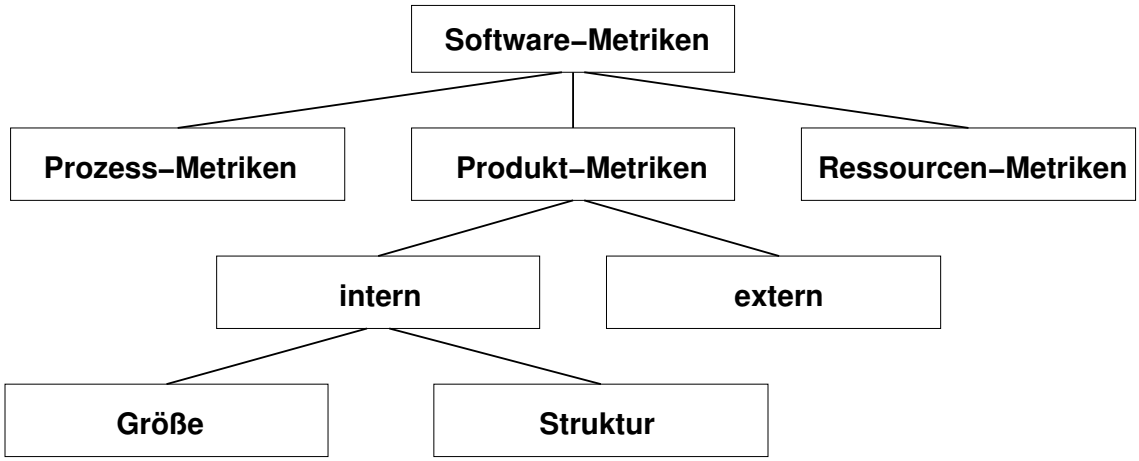
Software-Metrik

Definition

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

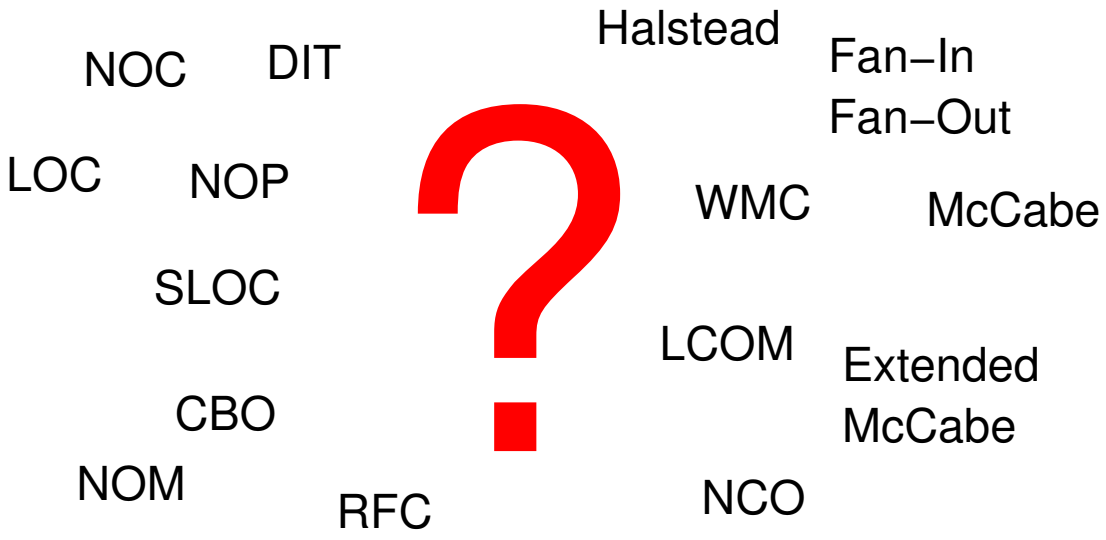
– IEEE Standard Glossary

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, ...

Metriken



... und alle möglichen Kombinationen, z.B. NOC/Package als *High-level Structuring*

Messen kann man viel. Aber wie findet man das Angemessene?

Größenmetriken:

- LOC - Lines of Code
- SLOC - Source Lines of Code (ohne Leerzeilen/Kommentare)
- Halstead - Größe unabhängig von Layout
- McCabe - Anzahl der Bedingungen + 1
- Extended - Wie McCabe aber mit Zählung von Operanden von `and` und `or`
- NOP - Number of Packages
- NOC - Number of Classes
- NOM - Number of Methods

Kopplungsmetriken

- Fan-In: Anzahl eingehender Abhängigkeiten
- Fan-Out: Anzahl ausgehender Abhängigkeiten

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

Größenmetriken – LOC

Lines of code (LOC)

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

Größenmetriken – LOC

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

Wieviel LOC?

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, ...

Größenmetriken – LOC

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

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


Größenmetriken – Halstead

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

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

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

Wie werden Operanden + Operatoren definiert?
z.B. Operator = Token, Operand = Literal und Bezeichner

Größenmetriken – weitere

weitere:

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

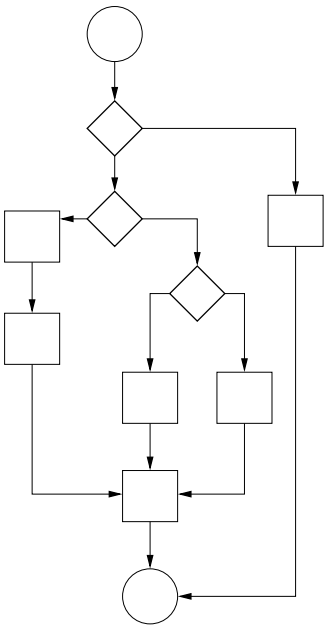
Strukturmetriken

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

Strukturmetriken – Kontrollflussgraph

Eigenschaften des Kontrollflussgraphen

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



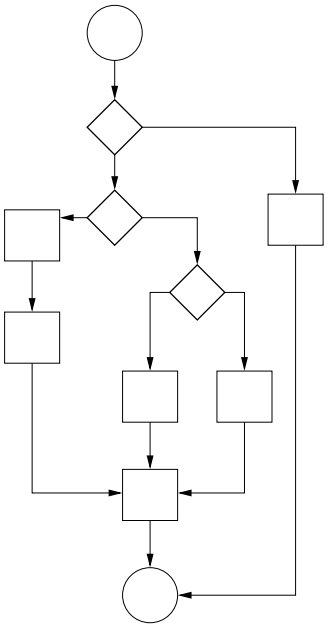
Komplexitätsmetriken – McCabe

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

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

oder einfacher:

$$V(g) = \#\text{Binärverzweigungen} + 1$$



+ einfach zu berechnen
– Komplexität von Anweisungen unberücksichtigt

[a] 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 $(i_1, i_2, \ldots, i_e)$ 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 $p_1, \ldots, p_n$, wenn es ganze Zahlen $a_1, \ldots, a_n$ gibt, so dass $p = \sum a_i p_i$ 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 Pfade 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;
    }
  }
}
```

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

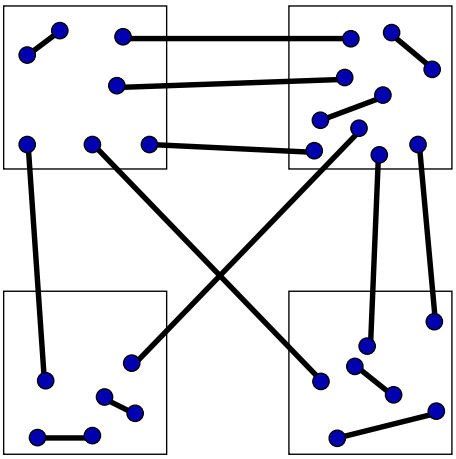
McCabe-Beispiele

```
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;
```

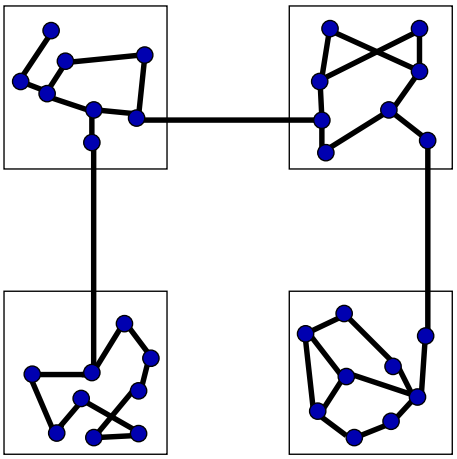
```
S : array (1..5) of Character := ('A', 'B', 'C', 'D', 'E');
i := 1;
loop
  exit when S(i) = A;
  i := i + 1;
end loop;
```

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


Strukturmetriken – Kopplung und Kohäsion



starke Kopplung
schwache Kohäsion



schwache Kopplung
starke Kohäsion

Strukturmetriken – OO

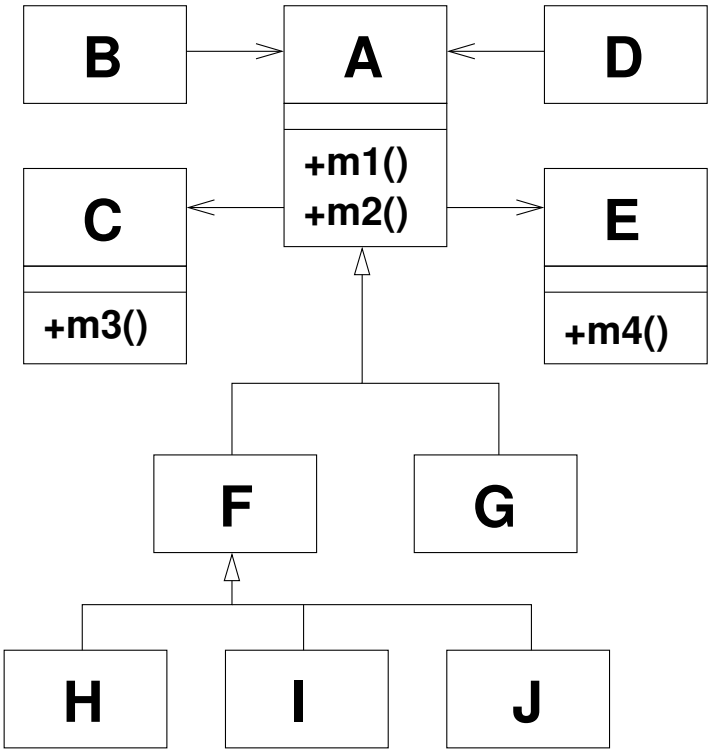
OO-Metriken (Chidamber 1994; Chidamber und Kemerer 1994):

- 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

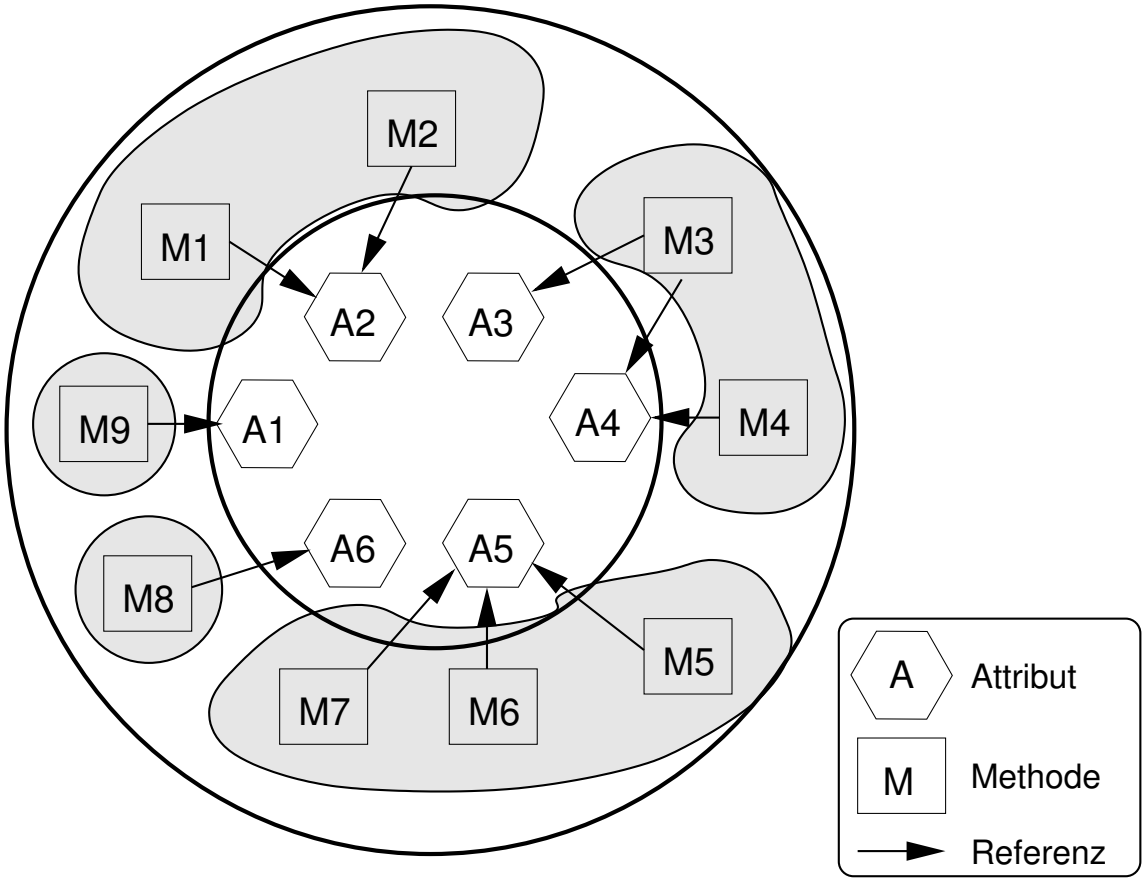
WMC: Anzahl Klassenmethoden, optional gewichtet nach Größe oder Komplexität
DIT: Länge des Weges von der Wurzel bis zur Klasse (tiefe Hierarchie ist fehleranfällig)
NOC: Anzahl direkter Unterklassen, hohe Zahl ist Indikator für gute Wiederverwendung
CBO: Anzahl Klassen, mit denen eine Klasse gekoppelt 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 - sum(mA)/(m*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

Strukturmetriken – OO



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

LCOM



Einsatz von Metriken

- Regelbasiertes Qualitätsmodell
- Benchmarking
- Visualisierung
  - einer Version
  - zeitlicher Verlauf
- Vorhersage von Qualitätseigenschaften

Regelbasierte Qualitätsmodelle

Beispiel

Gottklasse

- benutzt viele Attribute anderer Klassen
- hohe funktionale Komplexität
- geringe Kohärenz (innerer Zusammenhalt)

Metriken (Lanza und Marinescu 2006):

- *Access to Foreign Data (ATFD)*: Anzahl von Attributen anderer Klassen, die direkt oder mittels Zugriffsmethoden verwendet werden
- *Weighted Methods per Class (WMC)*: gewichtete Anzahl von Methoden
- *Tight Class Cohesion (TCC)*: relative Anzahl von Methodenpaaren einer Klasse, die gemeinsam auf mindestens ein Attribut derselben Klasse zugreifen

Regelbasierte Qualitätsmodelle

Feste Schwellwerte:

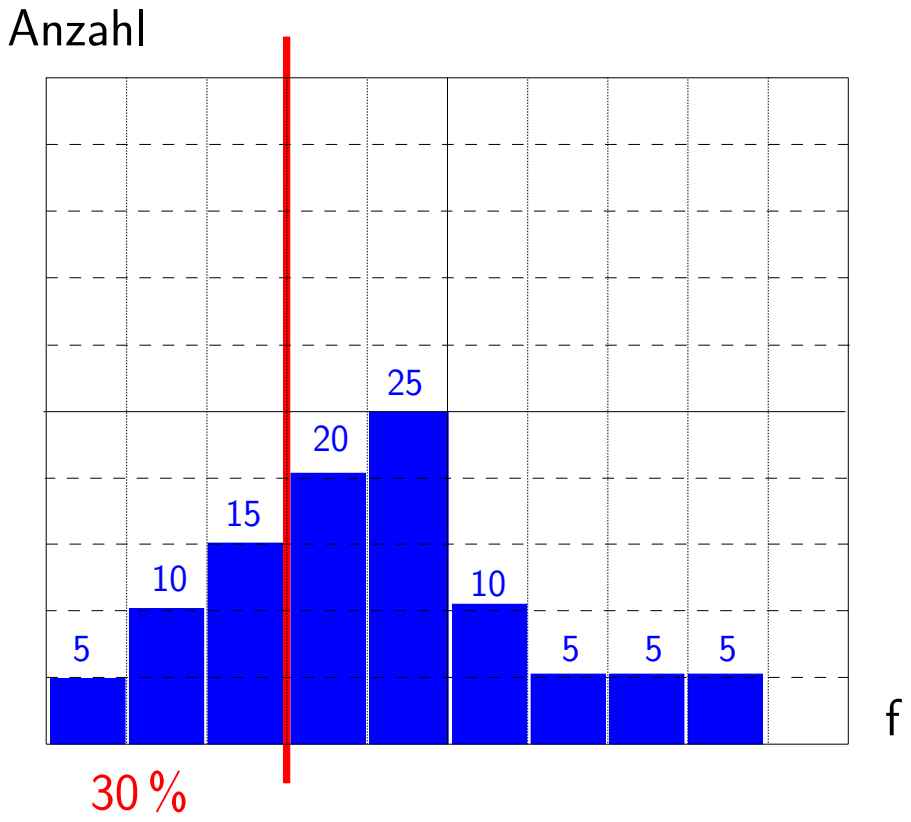
$$\text{ATFD} > 5 \wedge \text{WMC} \geq 800 \wedge 0 \leq \text{TCC} \leq 8$$

Relative und feste Schwellwerte:

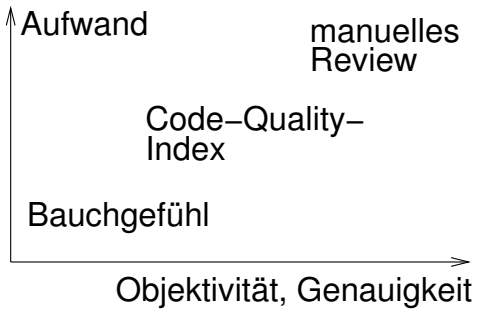
$$\text{ATFD} > \text{few} \wedge \text{WMC} \geq \text{very high} \wedge \text{TCC} \in P_{30}(\text{TCC})$$

- very high $= (AVG + STDEV) \times 1.5$
- few $= 5$
- Perzentil $P_n(f) =$ Menge der ersten $n\,\%$ Prozent der Elemente, die nach Werten von f geordnet sind

30-Perzentil P_{30}



Benchmarking: Code Quality Index (Simon u. a. 2006)



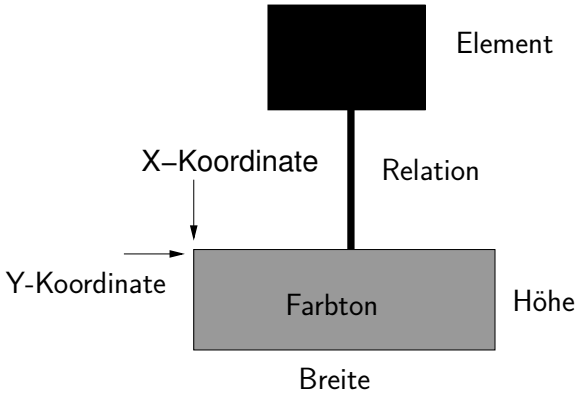
Quality-Benchmark-Level auf Basis eines statisch ermittelten, objektiven Code-Quality-Index:

- 52 Qualitätsindikatoren (Typen von Bad Smells)
- Häufigkeitsverteilung für mehr als 120 industrielle Systeme geschrieben in C++ und Java → „Industriestandard“

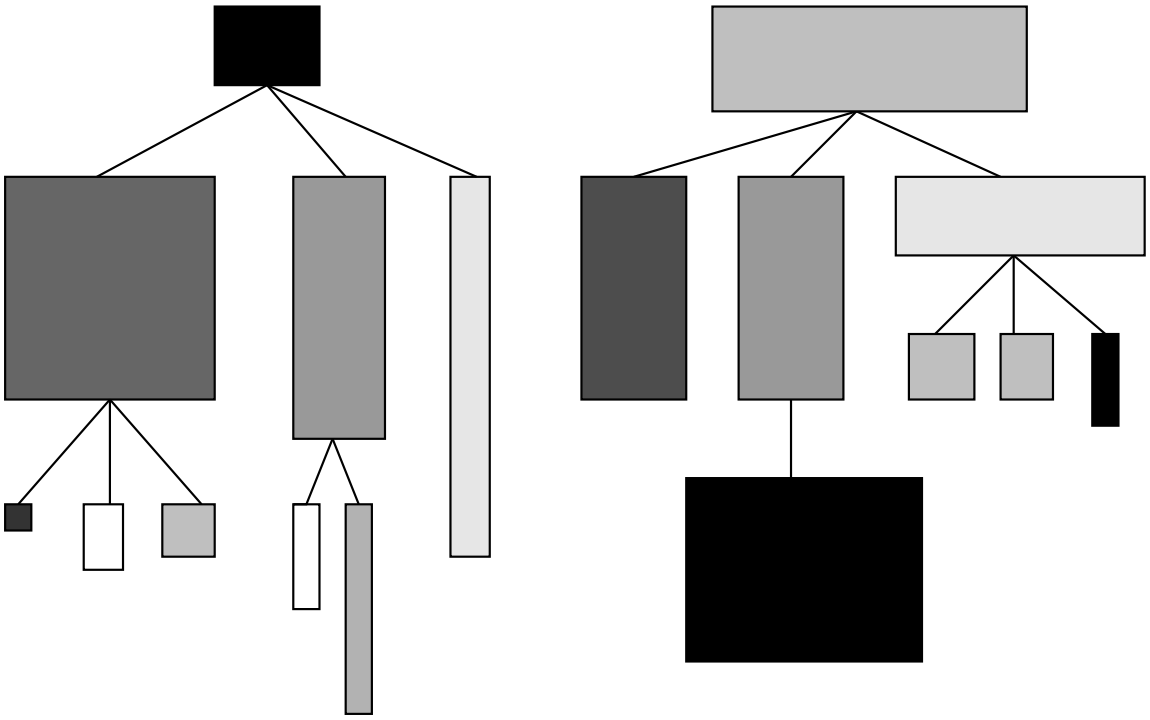
Visualisierung von Metriken (Lanza 2003)

Kombination von Metriken und Software-Visualisierung

- Graph-Repräsentation
- Bis zu fünf Metriken bestimmen die Visualisierung der Knoten:
  - Größe (1+2)
  - Farbe/Farbton (3)
  - Position (4+5)



Polymetrische Sichten

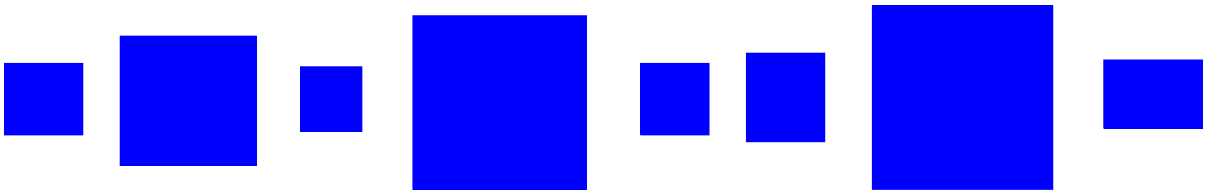


Polymetrische Sichten über die Zeit

Definition

Pulsar: wiederholte Änderungen, die Element größer und kleiner werden lassen.

→ System-Hotspot: Jede neue Version verlangt Anpassungen.

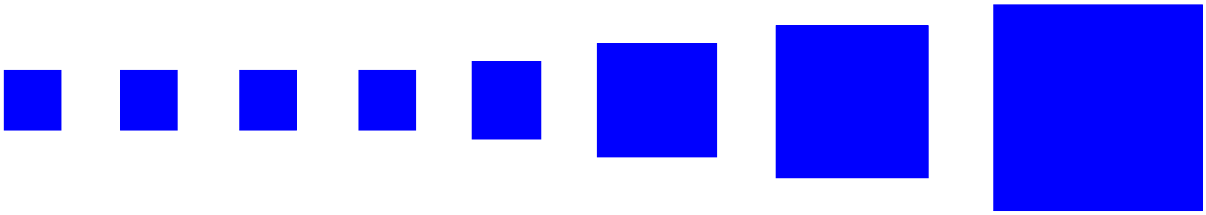


Polymetrische Sichten über die Zeit

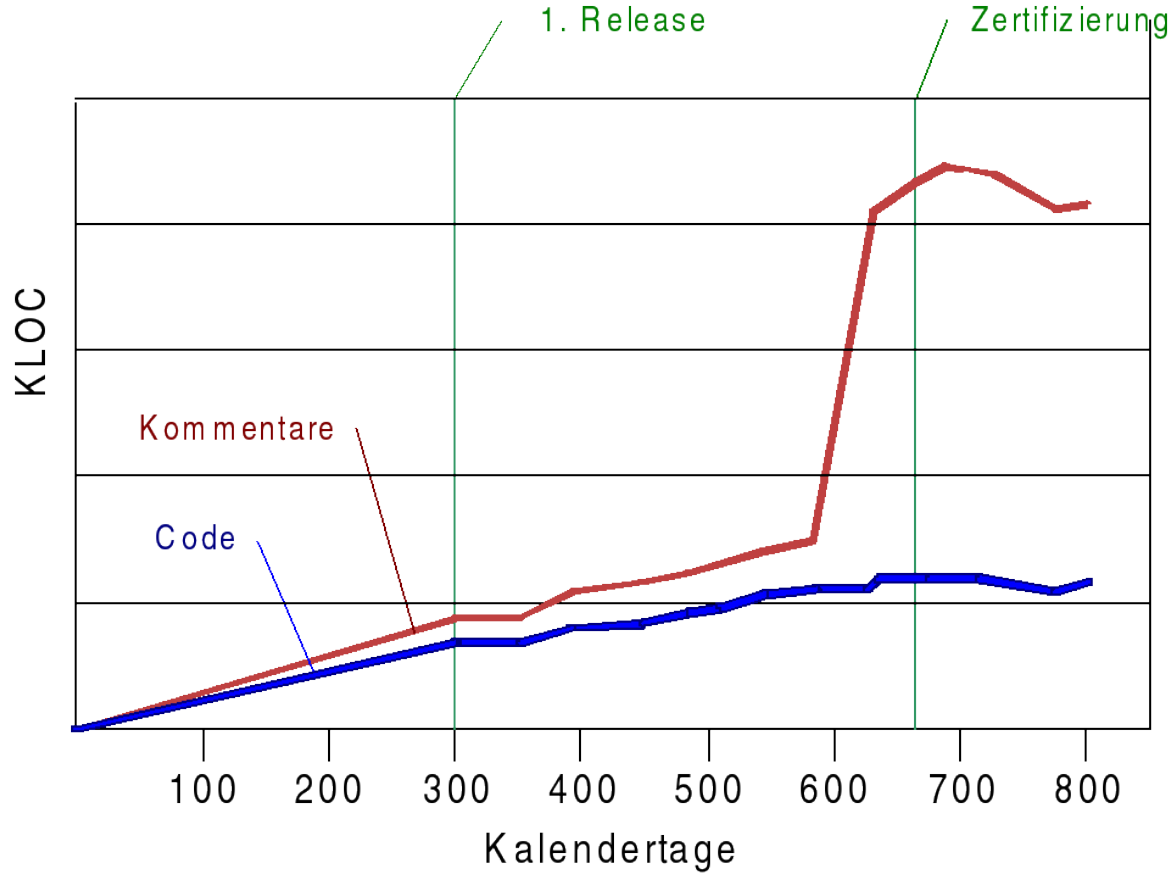
Definition

Supernova: Plötzlicher Anstieg. Mögliche Gründe:

- massive Restrukturierung
- Datenspeicher für Daten, die plötzlich hinzugekommen sind
- Schläfer: Stumpf, der mit Funktionalität gefüllt wird

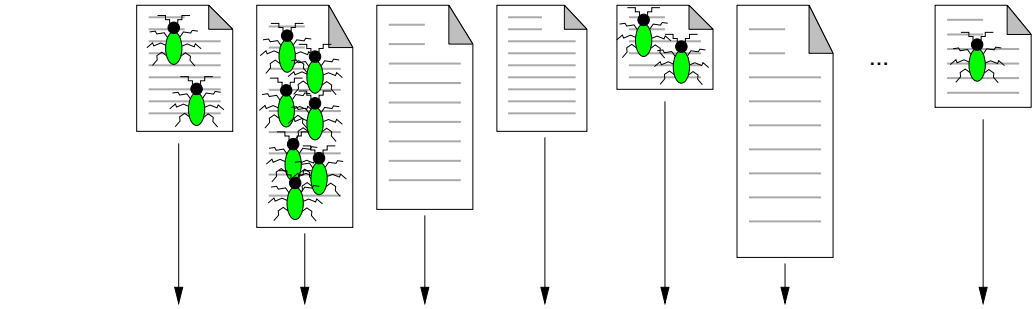


Trend-Analyse: Kommentierung

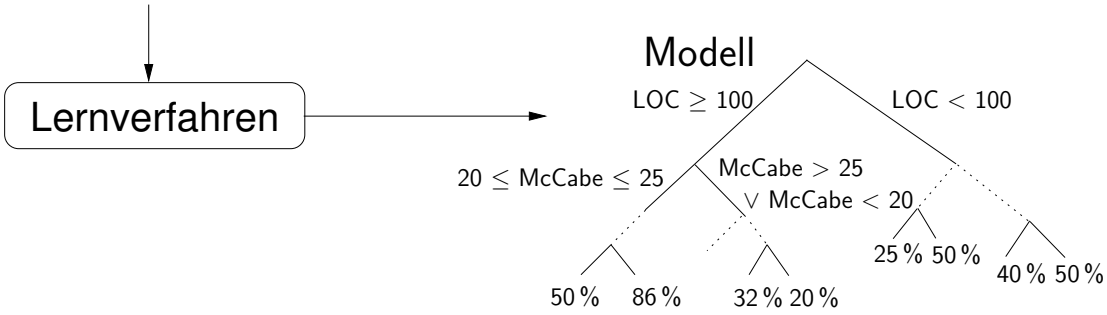


Visualisierung: Cockpits

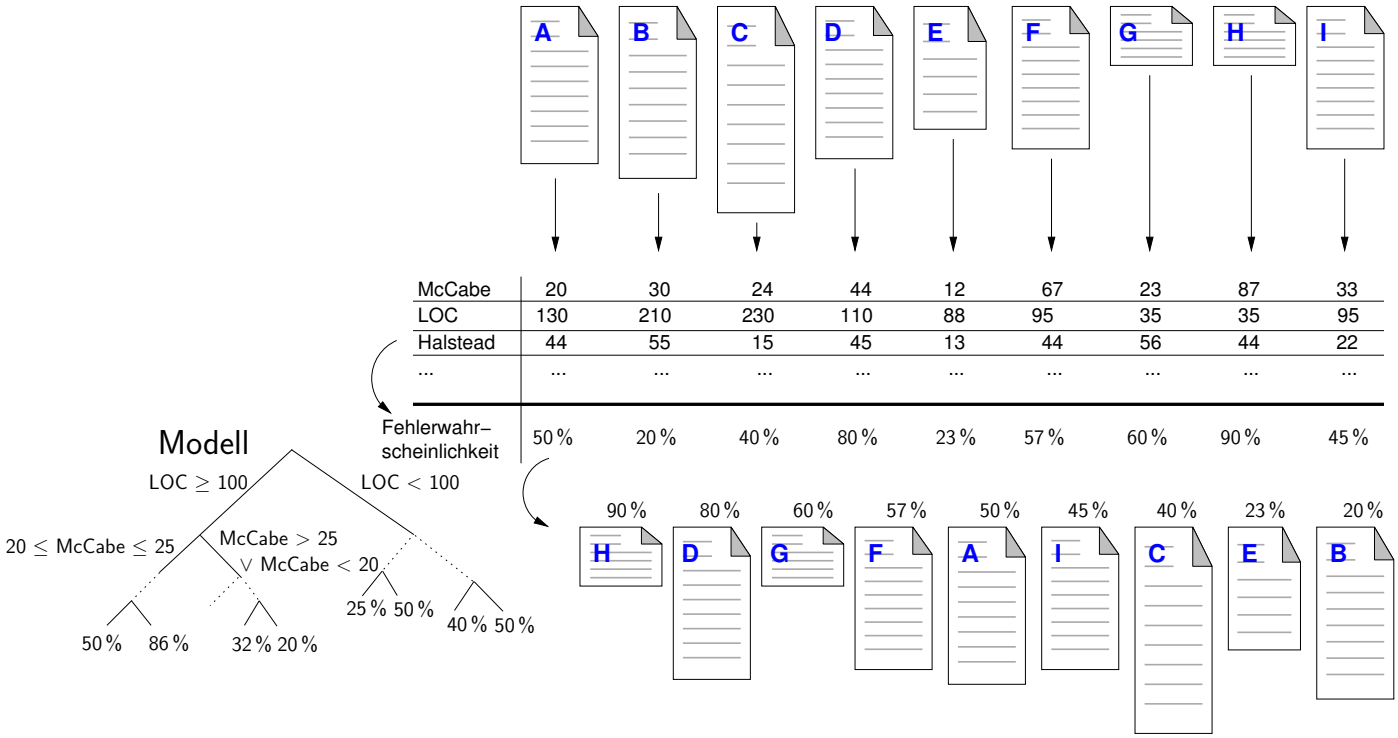
Qualitätseigenschaften: 1. Lernen



McCabe	20	25	45	29	22	13	...	19
LOC	100	200	180	100	77	222	...	85
Halstead	40	86	22	33	11	87	...	22
...	...	...	...	...	...	...	...	...
Fehler	ja	ja	nein	nein	ja	nein	...	ja



Qualitätseigenschaften: 2. Vorhersage



Vorhersage von Qualitätseigenschaften: Beispiele

Studie von Ostrand u. a. (2005):

- Metriken bewerten Module
- die ersten 20 % der Dateien der Bewertungsordnung enthalten 80 % der Fehler
- die ersten 20 % der Dateien der Bewertungsordnung enthalten 70 % des Codes

Studie von Mende und Koschke (2009):

- simple Ordnung nach Größe liefert vergleichbar gute Ergebnisse

Studie von El Emam u. a. (2001):

- OO-Metriken können Fehler vorhersagen
- OO-Metriken, die auf LOC normalisiert werden, können Fehler nicht vorhersagen

$\rightarrow$ Bewertung muss Test- und Fehlerfolgekosten einbeziehen (Mende und Koschke 2010)

Komplexitätsmetriken – McCabe

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 ≤ 15 als Qualitätsziel vor

SEI-Kategorisierung

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 Zugtunnels England-Frankreich schreibt für Prozeduren ZK ≤ 20 und LOC ≤ 50 vor (Bennett 1994).

Empirische Studien

Maintainability Index (Coleman/Oman, 1994):

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

- 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

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

Empirische Studien

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

Empirische Studien – OO

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

Empirische Studien – OO

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

Vorhersage von Qualitätseigenschaften

- es gibt viele Studien über Zusammenhänge von Metriken (Code, Änderungen, Entwickler, Checkin-Zeitpunkt, ...) und Qualitätseigenschaften
- die Untersuchungen haben kein einheitliches Bild geboten
- Vorsicht ist bei den Details der Evaluation angebracht
- dennoch werden Metriken in der Praxis benutzt (z.B. Code-Quality-Index, Wartbarkeitssiegel von TÜV/Nord)

Metriken: Grenzen

- Metriken sind Information, nicht Wissen
- Metriken müssen interpretiert werden
- Metriken sind starke Vereinfachungen, Details gehen verloren
- jede Metrik kann unterlaufen werden

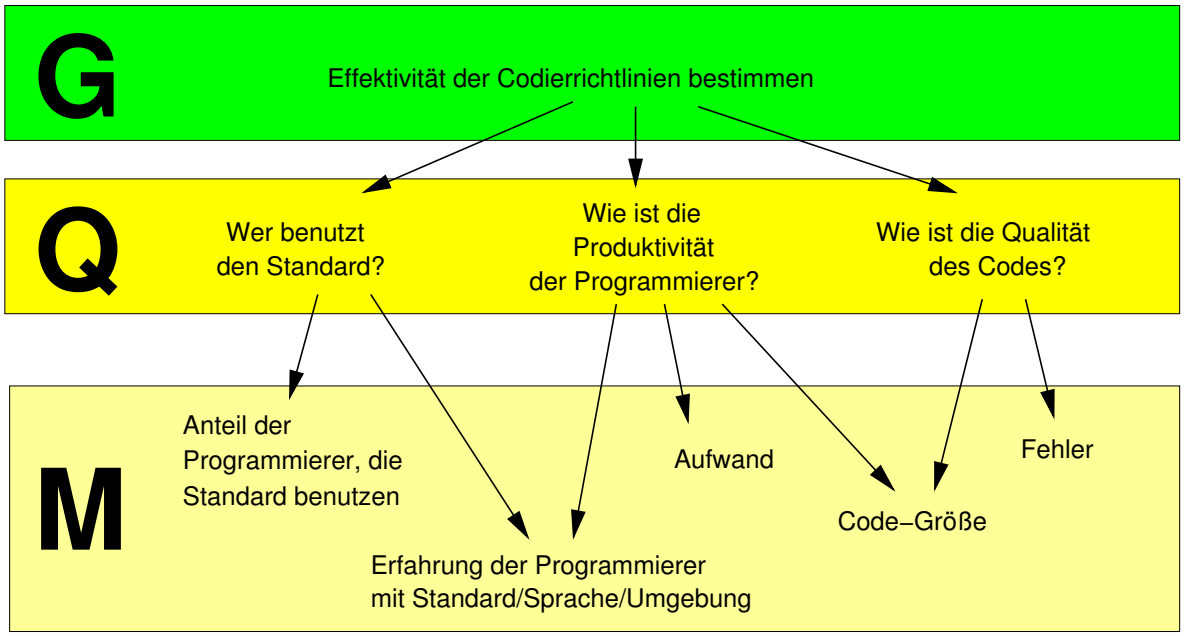
Metriken richtig einsetzen: Zielorientiertes Messen

GQM (Goal-Question-Metric) nach Basili und Weiss (1984):

Nicht das messen, was einfach zu bekommen ist,
sondern das, was benötigt wird

1. Ziele erfassen.
2. Zur Prüfung der Zielerreichung notwendige Fragen ableiten.
3. Was muss gemessen werden, um diese Fragen zu beantworten?

Zielorientiertes Messen



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, ...

Wiederholungs- und Vertiefungsfragen I

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

1 Basili und Weiss 1984 Basili, R. ; Weiss, D. M.: A Methodology for Collecting Valid Software Engineering Data. In: IEEE Computer Society Transactions on Software Engineering 10 (1984), November, Nr. 6, S. 728–738

2 Bennett 1994 Bennett, P.A.: Software Development for the Channel Tunnel: a Summary. In: High Integrity Systems 1 (1994), Nr. 2, S. 213–220

3 Bruntink und van Deursen 2004 Bruntink, M. ; Deursen, A. van: Predicting Class Testability Using Object-Oriented Metrics. In: IEEE International Workshop on Source Code Analysis and Manipulation, IEEE Computer Society Press, September 2004

4 Chidamber 1994 Chidamber, S.: Metrics Suite For Object Oriented Design, M.I.T, Cambridge, Dissertation, 1994

Software-Metriken: Objektorientierte Metriken

5 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), Juni, Nr. 6, S. 476–493

6 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

7 El Emam u. a. 2001 El Emam, Kalhed ; Benlarbi, Saïda ; Goel, Nishith ; Rai, Shesh N.: The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics. In: IEEE Computer Society Transactions on Software Engineering 27 (2001), Nr. 7, S. 630–650. – ISSN 0098-5589

8 Fenton und Pfleeger 1996 Fenton, N. ; Pfleeger, S.: Software Metrics: A Rigorous and Practical Approach. 2nd. London : International Thomson Computer Press, 1996

9 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

10 Grady 1994 Grady, R.B.: Successfully Applying Software Metrics. In: IEEE Computer 27 (1994), September, Nr. 9, S. 18–25

11 Lanza 2003 Lanza, Michele: Object-Oriented Reverse Engineering - Coarse-grained, Fine-grained, and Evolutionary Software Visualization. `http://www.inf.unisi.ch/faculty/lanza/Downloads/Lanz03b.pdf`, University of Bern, Dissertation, 2003

12 Lanza und Marinescu 2006 Lanza, Michele ; Marinescu, Radu: 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

Software-Metriken: Objektorientierte Metriken

13 Mende und Koschke 2009 Mende, Thilo ; Koschke, Rainer: Revisiting the Evaluation of Defect Prediction Models. In: PROMISE '09: Proceedings of the 5th International Conference on Predictor Models in Software Engineering. New York, NY, USA : ACM, 2009, S. 1–10. – ISBN 978-1-60558-634-2

14 Mende und Koschke 2010 Mende, Thilo ; Koschke, Rainer: Effort-Aware Defect Prediction Models. In: European Conference on Software Maintenance and Reengineering, 2010. – submitted for publication

15 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

16 Ostrand u. a. 2005 Ostrand, T.J. ; Weyuker, E.J. ; Bell, R.M.: Predicting the location and number of faults in large software systems. In: IEEE Computer Society Transactions on Software Engineering 31 (2005), Nr. 4, S. 340–355. – ISSN 0098-5589

17 Simon u. a. 2006 Simon, Frank ; Seng, Olaf ; Mohnhaupt, Thomas: Code-Quality-Management – Technische Qualität industrieller Softwaresysteme transparent und vergleichbar gemacht. dpunkt.verlag, 2006