Softwaremetriken

Alexander Pilch (520376) Kai Wetzelsberger (520553)

Fakultät für Informatik
Hochschule Mannheim

29. Juni 2006

Inhalt

Motivation

Fred S. Roberts (1979) [ZH11995]

A major difference between a „well developed“ science such as physics and some of the less „well developed“ sciences such as psychology or sociology is the degree to which things are measured

Definition nach IEEE Standard 1061

„Eine Softwarequalitätsmetrik ist eine Funktion, die eine Software-Einheit in einen Zahlenwert abbildet. Dieser berechnete Wert ist interpretierbar als der Erfüllungsgrad einer Qualitätseigenschaft der Software-Einheit“

Definition nach Ian Sommerville („Software Engineering“)

„Eine Softwaremetrik ist jede Art von Messung, die sich auf ein Softwaresystem, einen Prozess oder die dazugehörige Dokumentation bezieht“

Historie (I) [ZH11995]

- **1960er** SLOC (**S**ource **L**ines **o**f **C**ode)
  ⇒ Anzahl der Lochkarten (max. 80 Zeichen pro Karte)
- **1968** Rubey: erste Veröffentlichung über Softwarekomplexität
  ⇒ Was ist Softwarekomplexität ? Ist sie meßbar ?
- **1975** Kolence: Entstehung des Begriffs „Software Physics“
- **1976** McCabe: Zyklomatische Komplexität
  ⇒ Ziel: Kontrolle der Programmgröße durch Vorgabe einer oberen Schranke
- **1977** Gilb: Veröffentlichung eines der ersten Bücher über Metriken
- **1977** Halstead: Halstead Metriken

Historie (II) [ZH11995]

- **1979** Albert: Function Point Metrik
- **1981** Studie über die Aussagekraft von Metriken
- **1981** Böhm: COCOMO (**Co**nstructive **Co**st **Mo**del)
  $\Rightarrow$ Auf LOC basierend
- **1983** Basili und Hutchens: LOC nicht ausreichend
- **1984** NASA und SEI beginnen den Einsatz von Softwaremetriken
- **1984** Entstehung von GQM (**G**oal-**Q**uestion-**M**etric)
- **1989** Diskussion von Softwaremetriken in Objektorientierung
- **1992** IEEE 1061: Software Quality Metric Standard
- **1994** Chidamber und Kemerer: OO-Metriken

Ziele der Entwicklung von Metriken [ZH11995]

- Vorhersage von Fehlern in der Designphase
- Aussagen über zukünftige Wartbarkeit, Testbarkeit (z.B. McCabe)
- Aufwandsabschätzung (z.B. Function Points)
- Reduzierung von Informationen auf eine Zahl (z.B. Komplexität)
  $\Rightarrow$ Vereinfachte Bewertung von Qualitätseigenschaften

Fehlinterpretation (I) - Simpson-Paradoxon [ZH21995]

	Abteilung 1	Abteilung 2
1980	Frauen : 50 Männer : 50 50% Frauenanteil	Frauen : 5 Männer : 25 17% Frauenanteil
1990	Frauen : 15 Männer : 10 60% Frauenanteil	Frauen : 20 Männer : 75 21% Frauenanteil

Fehlinterpretation (II) - Simpson-Paradoxon [ZH21995]

	1980	1990
gesamt	Frauen : 55 Männer : 75 42% Frauenanteil	Frauen : 35 Männer : 85 29% Frauenanteil

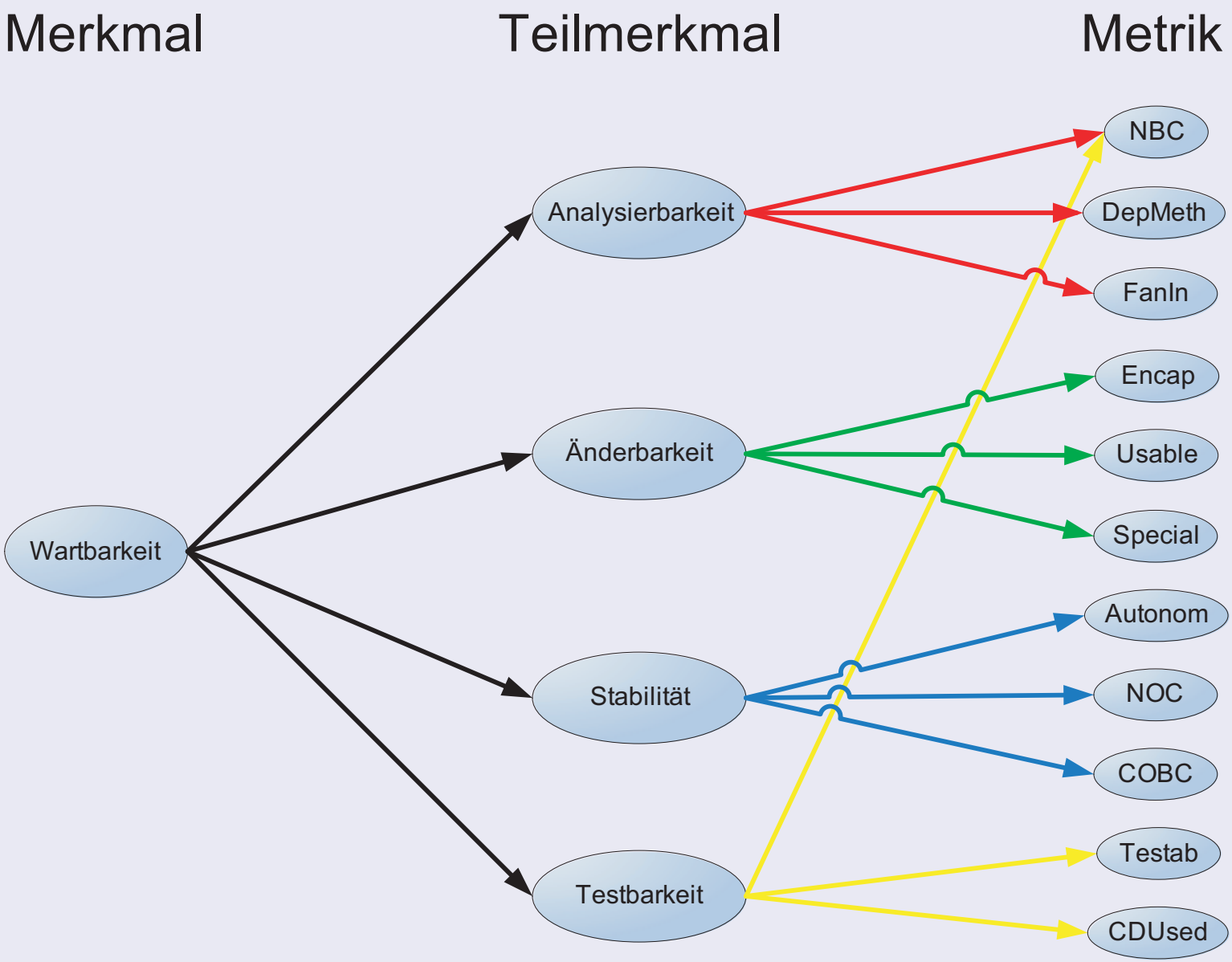
Defect-Density Metrik

$$DD = \frac{Defects}{(K)LOC}$$

Konsequenzen [ZH1999]

- Metrik verwendet keine Verhältnisskala
  $\Rightarrow$ Aussagen über kumulierte Werte nicht möglich
- „Intuitive“ Operationen können zu fehlerhaften Aussagen führen

Factor Criteria Model (FCM)



Untersuchung zum Metrikeinsatz in der Industrie [METR2005]

Fragestellung

- Welche Metriken werden tatsächlich in der Praxis erhoben?
- Wer erhebt und wer verwendet Metriken?
- Welche Probleme werden mit diesen Metriken angegangen?
- Welcher Nutzen wird aus dem Metrikeinsatz gezogen?

Beteiligte

- 21 Unternehmen aus unterschiedlichen Branchen in Deutschland, vorwiegend Großunternehmen aus der Automobil-, IT- sowie der Versicherungs- und Bankenbranche.
- 13 Unternehmen zertifiziert nach CMM (5) und / oder ISO-9000 (11)
- Befragte als Projektleiter, Berater, Entwickler, im Management oder in der Qualitätssicherung tätig.

Arbeitsbereiche der Beteiligten

- Benchmarking
- Prozessverbesserung
- Softwaremetriken
- Qualitätsmanagement
- Anwendungsentwicklung
- Technisches Produktmanagement
- Design und Entwicklung
- Entwickler (2)

Produktklassen

- Anwendungen (13)
- Eingebettete Systeme (7)
- Systemsoftware (6)

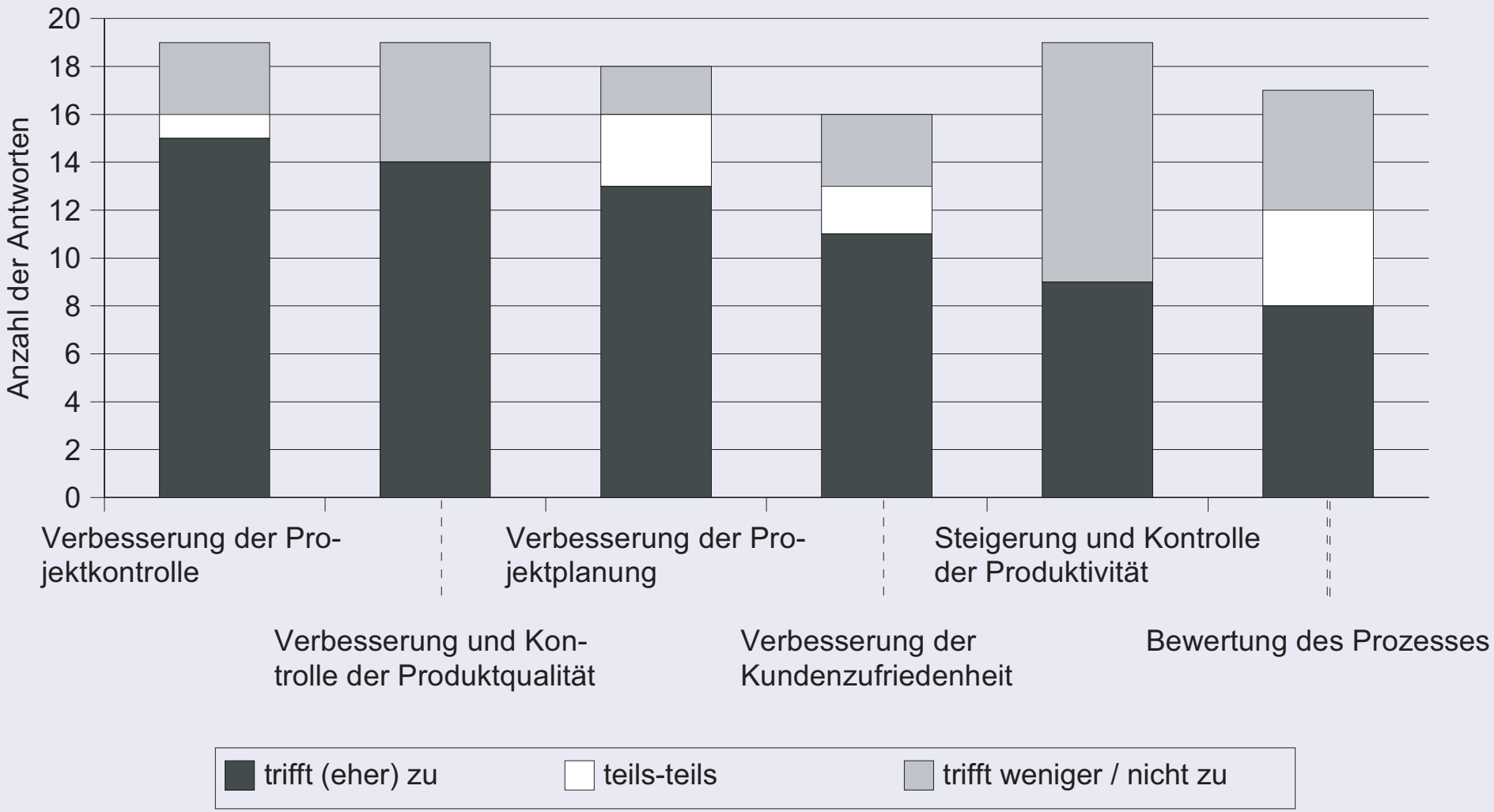
1. Phase - Fragebogen

- Bestehend aus halboffenen Fragen
- Ergänzungen, Mehrfachantworten und „keine Angaben“ waren möglich

2. Phase - Interviews

- Detaillierte Interviews mit ausgewählten Personen

Angestrebte Ziele für den Einsatz



Größenmetriken (14)

- Anzahl der Anforderungen (10)
- Anzahl der Komponenten (6)
- Lines of Code (10)

Einschätzung / Ziele

- Ableitungen von Aussagen bezüglich des Arbeitsfortschritts
- Kostenschätzung
- Fehlerabschätzung und -prognose
- Abhängig vom Arbeitsstil der einzelnen Mitarbeiter
- Sehr leicht zu unterlaufen

Komplexitätsmetriken (4)

- McCabe zyklomatische Komplexität (2)
- Schachtelungstiefe (2)
- Kopplung (2)
- Zusammenhalt (2)

Einschätzung / Ziele

- Prognose und Bewertung der Wartbarkeit
- Bessere Codequalität
- Bewertung der Programmkomplexität
- Schwer zu interpretieren
- (Un-) bewusste Verfälschung durch Mitarbeiter möglich

Objektorientierte Metriken (2)

Chidamber und Kemerer (2)

- Weighted Methods per Class (1)
- Depth in inheritance tree (1)
- Number of children (1)
- Coupling between objects (1)
- Lack of cohesion on methods (2)

Ziele

- Prognose und Bewertung der Wartbarkeit
- Verbesserung des Entwurfs
- Nachweis von Strukturverbesserungen (z.B. durch Refactoring)
- Ergänzung zu den Größen- und Komplexitätsmetriken

Metriken in der Qualitätssicherung (15)

- Anzahl der in Reviews entdeckten Fehler (9)
- Anzahl der in Tests entdeckten Fehler (13)
- Anzahl der Testfälle (11)

Einschätzung / Ziele

- Prüfmaßnahmen
- Prognosen über Zuverlässigkeit der Software
- Kostensenkung
- Festlegung des weiteren Testvorgehens
- Kontrolle des Prozesses

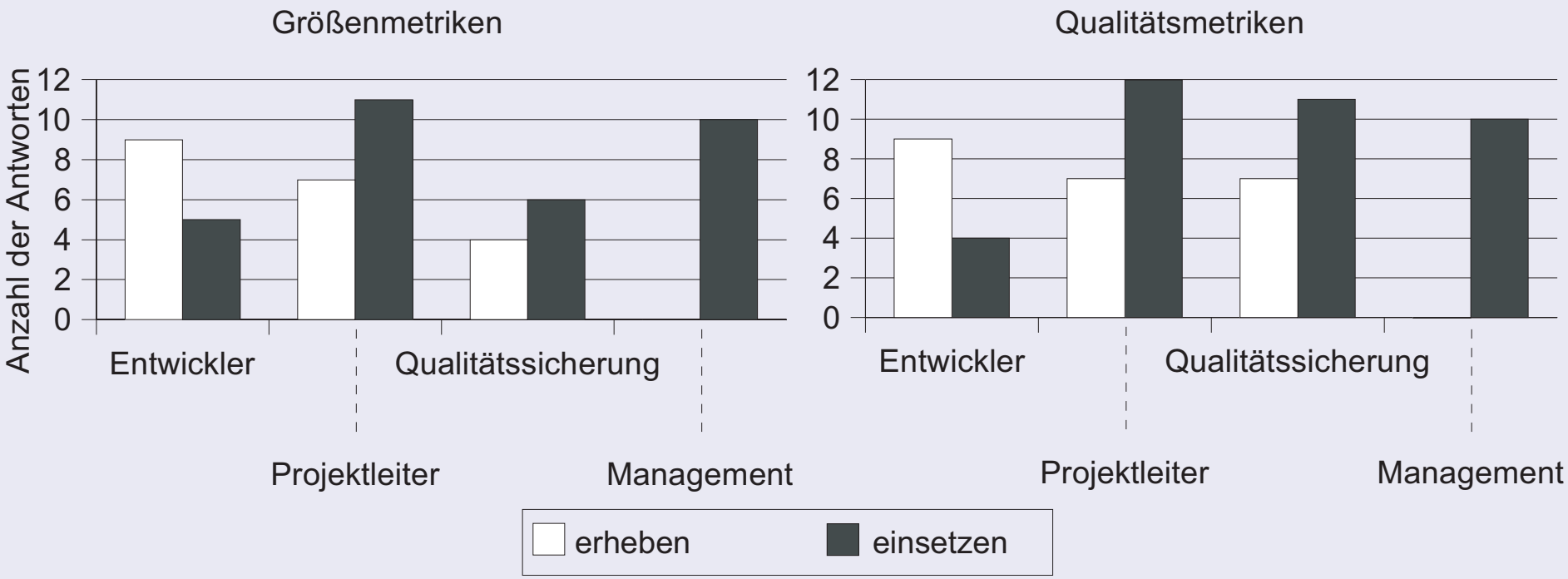
Metriken in der Wartung (17)

- Dauer (9)
- Aufwand (10)
- Anzahl der Problemmeldungen (10)
- Änderungswünsche (11)

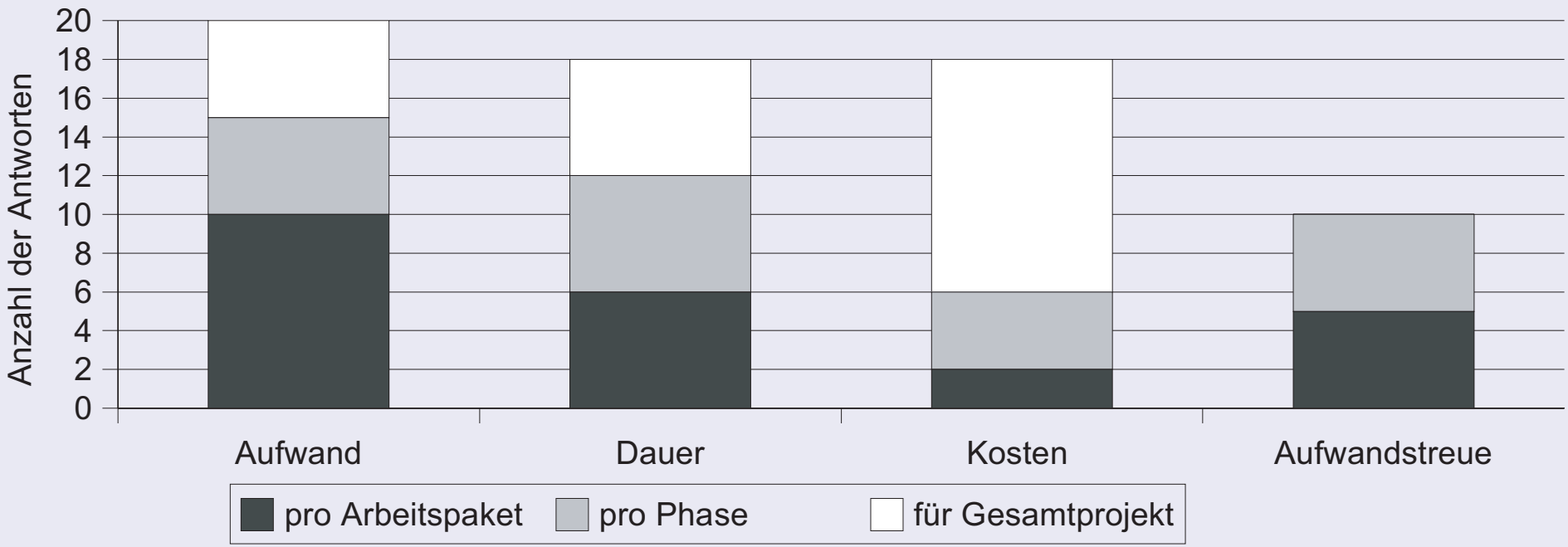
Einschätzung / Ziele

- Planung und Kontrolle der Wartungsprojekte
- Ermittelung der Wartungskosten
- Einhaltung von Wartungsverträgen

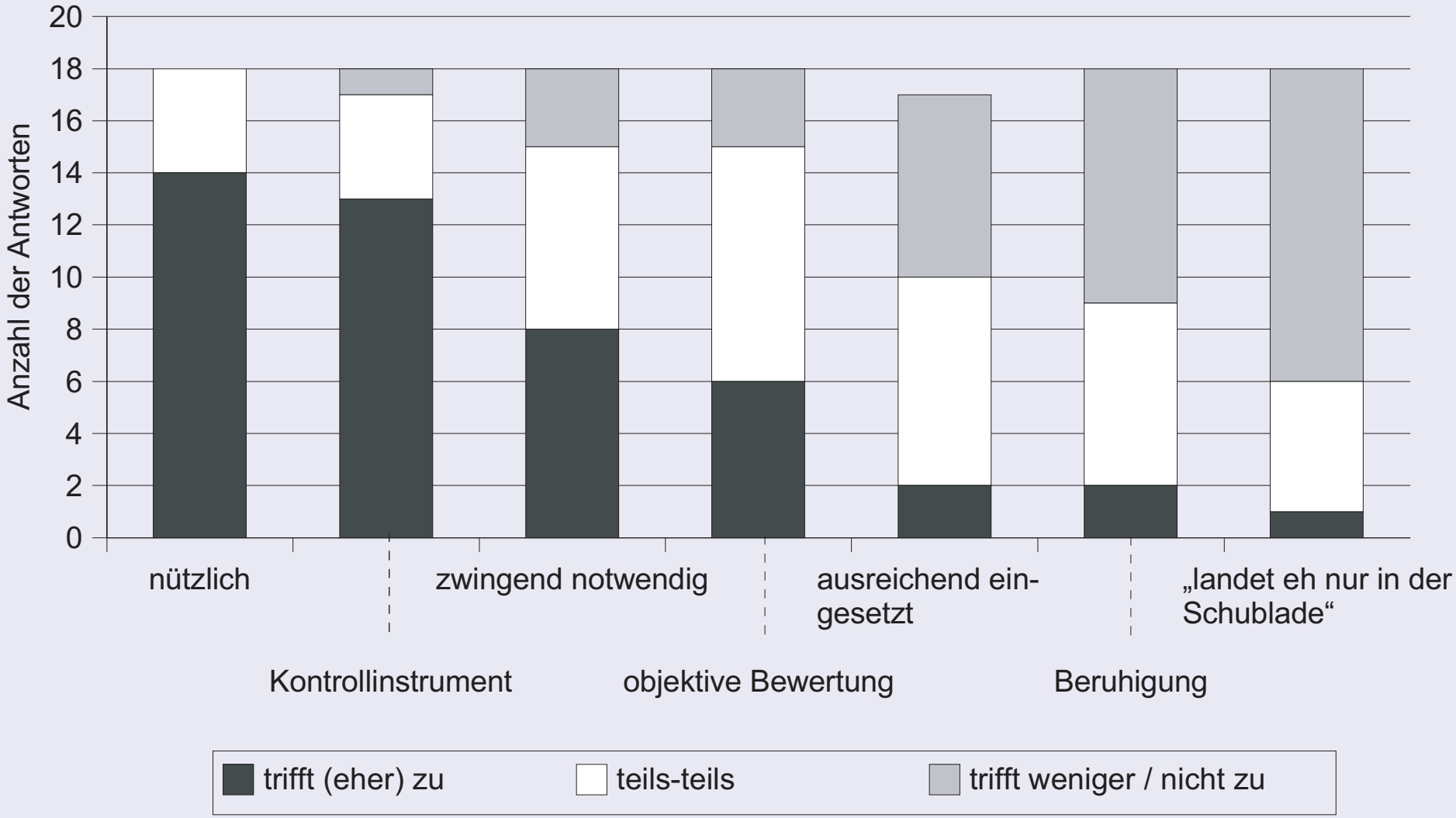
Einsatz von Größen- und Qualitätsmetriken



Einsatz in der Planung und Steuerung von Projekten



Ansichten über Metriken



Kritikpunkte

- Geringe Personenzahl (21 Personen)
- Nicht ausgewogene Wahl der Teilnehmer (nur 2 Entwickler)
  $\Rightarrow$ Vorhersagbarkeit der Ergebnisse
- Beinflussung der Ergebnisse durch die Auswahl der Beteiligten durch Ansprechpartner in den Firmen
- Ab CMM-Level 4 werden Metriken vorgeschrieben, jedoch findet keine weitere Unterteilung der Befragten statt.

McCabe zyklomatische Komplexität

Was ist das ?

- Maß für die Komplexität einer Funktion / eines Moduls
- 1976 von Thomas McCabe entwickelt
- Sprachenunabhängige Metrik

Berechnung [TJM1976]

$$V(g) = E - N + 2 \cdot P$$

Variablen

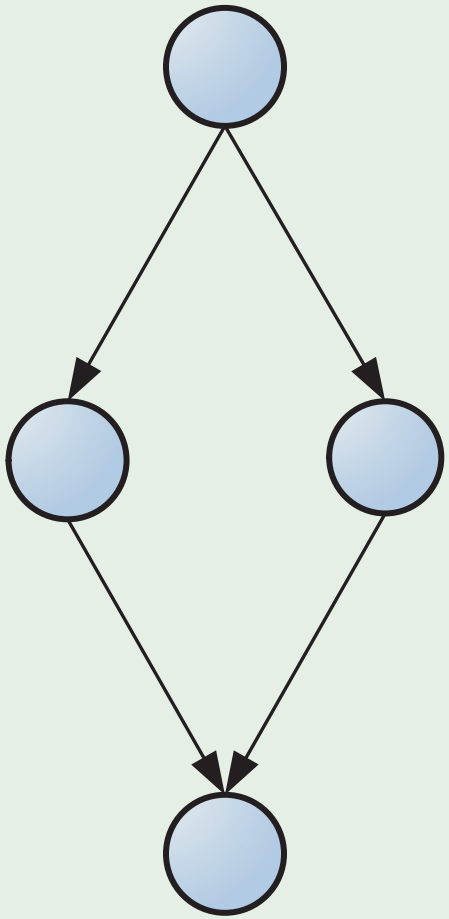
- E: Anzahl Kanten des Graphen
- N: Anzahl Knoten des Graphen
- P: Anzahl verbundener Komponenten / Module
  - Falls nur 1 Modul betrachtet wird entspricht $V(g)$ der Anzahl an Kontrollflüssen
- Ein linearer Kontrollfluß besitzt die Komplexität 1.
- Je größer $V(g)$ ist, desto mehr weicht der Kontrollfluss vom linearen ab.

Kontrollstrukturen

Sequenz

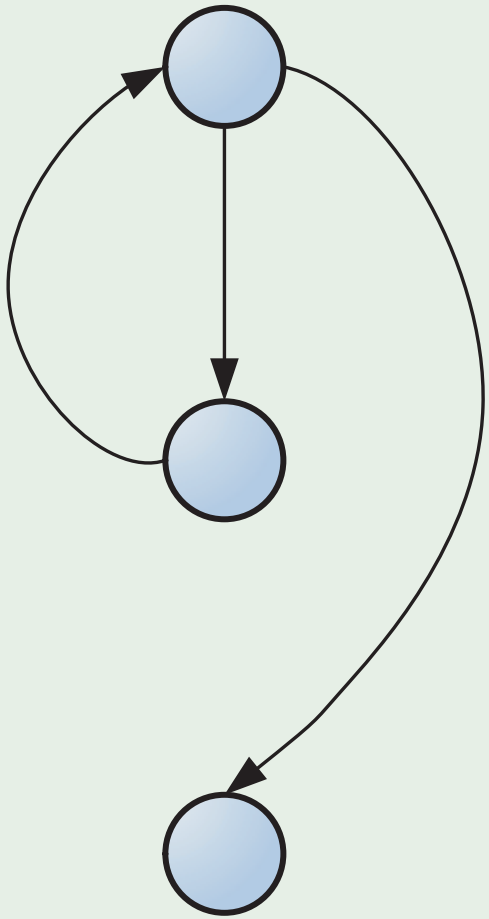
$V(g) = 1 - 2 + 2 = 1$

Auswahl



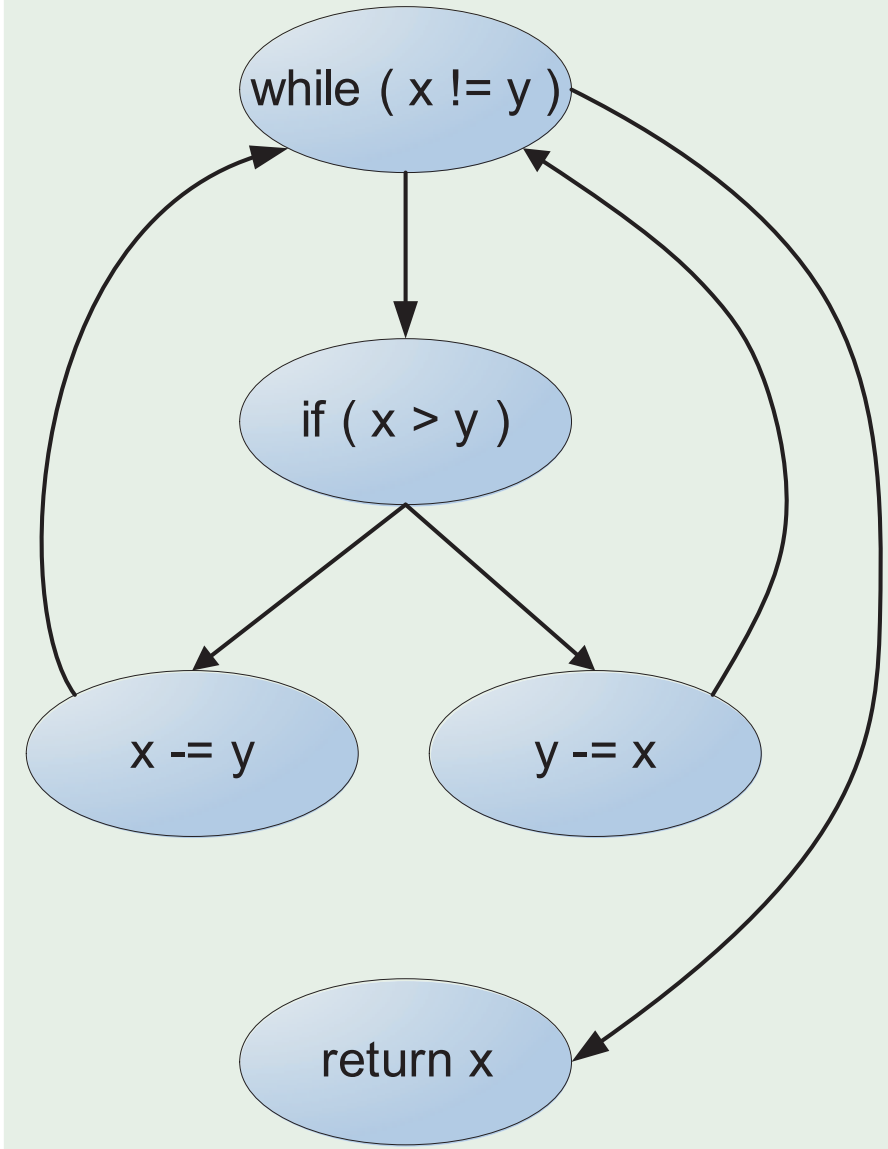
$V(g) = 4 - 4 + 2 = 2$

Schleife



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

ggT



Variablen

- Kanten: $E = 6$
- Knoten: $N = 5$
- Module: $P = 1$

Berechnung

$$V(g) = 6 - 5 + 2 \cdot 1 = 3$$

Bedingungen

if & while

Einsatzgebiete

- Risikoanalyse während der Entwicklung
  - Anpassung der Implementierung
- Risikoanalyse in der Wartung („code rot“)
  - Minimierung der Komplexität bei Änderungen am Code (Vergleich vor / nach Änderungen)
- Testplanung / Umfang von Tests
  - Bedingungsabdeckung
- Reengineering
  - Kosten- und Risikenabschätzung

Vorteile

- Einfach zu berechnen
- Geeignet zur Findung von Testfällen
- Grobes Maß für die Kontrollflusskomplexität
- Studien zeigen: generell gute Korrelation zwischen zyklomatischer Zahl und Verständlichkeit der Komponente

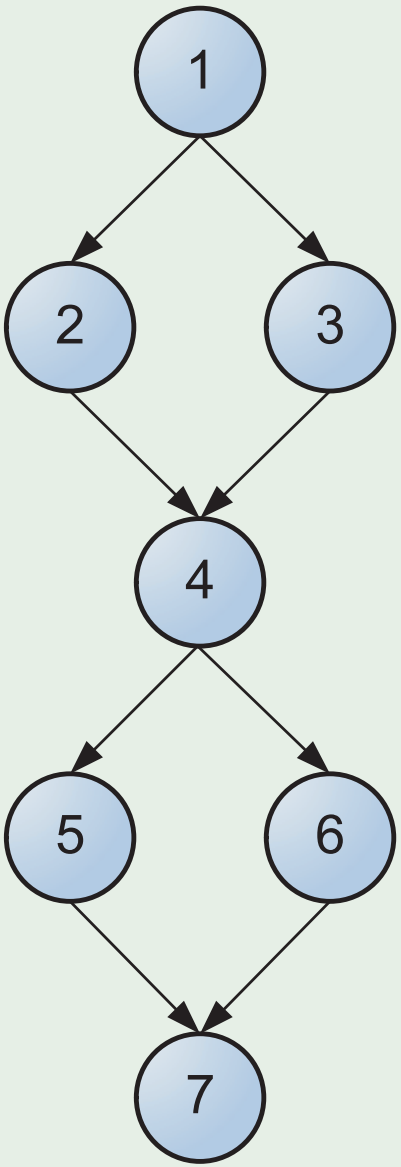
Kritikpunkte

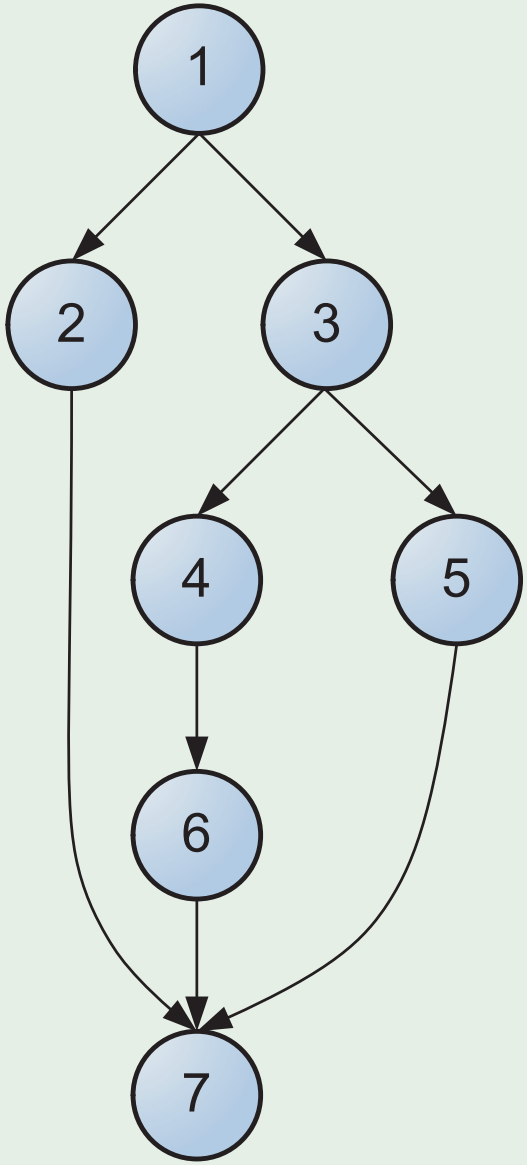
- Komplexität wird auf eine Zahl reduziert
  $\Rightarrow$ „zu einfach“
- Jede Verzweigung / Bedingung im Code wird gleich bewertet
- Für Funktionen entwickelt. Berücksichtigt nicht OO (Abstraktion, Vererbung, Kapselung, Polymorphismus, ...)
- Komplexität der Anweisungen wird nicht berücksichtigt
- Komplexität des Datenflusses wird nicht berücksichtigt

Anmerkungen

- Hohe zyklomatische Komplexität weist nicht unbedingt auf schlechten / unwartbaren Code hin z.B. viele einzelne if-else Anweisungen ggü. geschachtelten if-else Konstrukten
  - Hohe identische zyklomatische Komplexität, durch Anzahl an Bedingungen
- Niedrige Komplexität bedeutet nicht direkt guten Code z.B. Eine Bedingung bestehend aus 10 Prädikaten
  - Niedrige Komplexität, da nur eine Bedingung
- Softwaretools ignorieren teilweise die Anzahl der Module, da die Komplexität nur noch auf Methodenlevel berechnet wird.

$V(g) = 8 - 7 + 2 = 3$





Sequentielle Bedingungen

```
 1 | if (1) {
 2 |    2;
 3 | } else {
 4 |    3;
 5 | }
 6 |
 7 | if (4) {
 8 |    5;
 9 | } else {
10 |    6;
11 | }
12 |
13 | 7;
```

Geschachtelte Bedingungen

```
 1 | if (1) {
 2 |    2;
 3 | } else if (3) {
 4 |    4;
 5 |    6;
 6 | } else {
 7 |    5;
 8 | }
 9 |
10 | 7;
```

NASA Metrics Data Program [NAS2006]

Was ist das ?

- Repository der NASA mit freiem Zugang
- Datenbasis für eigene Auswertungen

Welche Daten sind verfügbar ?

- $>$ 2700 protokollierte Fehler in über 8 Jahren
- 27 Produktmetriken
- 10 Klassenmetriken
- 9 Anforderungsmetriken

Projekt JM1 [NAS2006]

This is a real time C project which has approximately 315 KSLOC.
There are eight years of error data associated with the metrics.
The changes to the modules are based on the changes reported within the problem reports.

Auszug Testdaten

MODULE	CYCLOMATIC_COMPLEXITY
11181	470
11182	128
11183	268
11184	19
11185	404
11186	127
11187	263
11188	94
11189	207
11190	42
11191	24
11192	94
11193	34
11194	286

Datensatz

- Projektdaten in CSV-Form abrufbar
- Anonymisiert

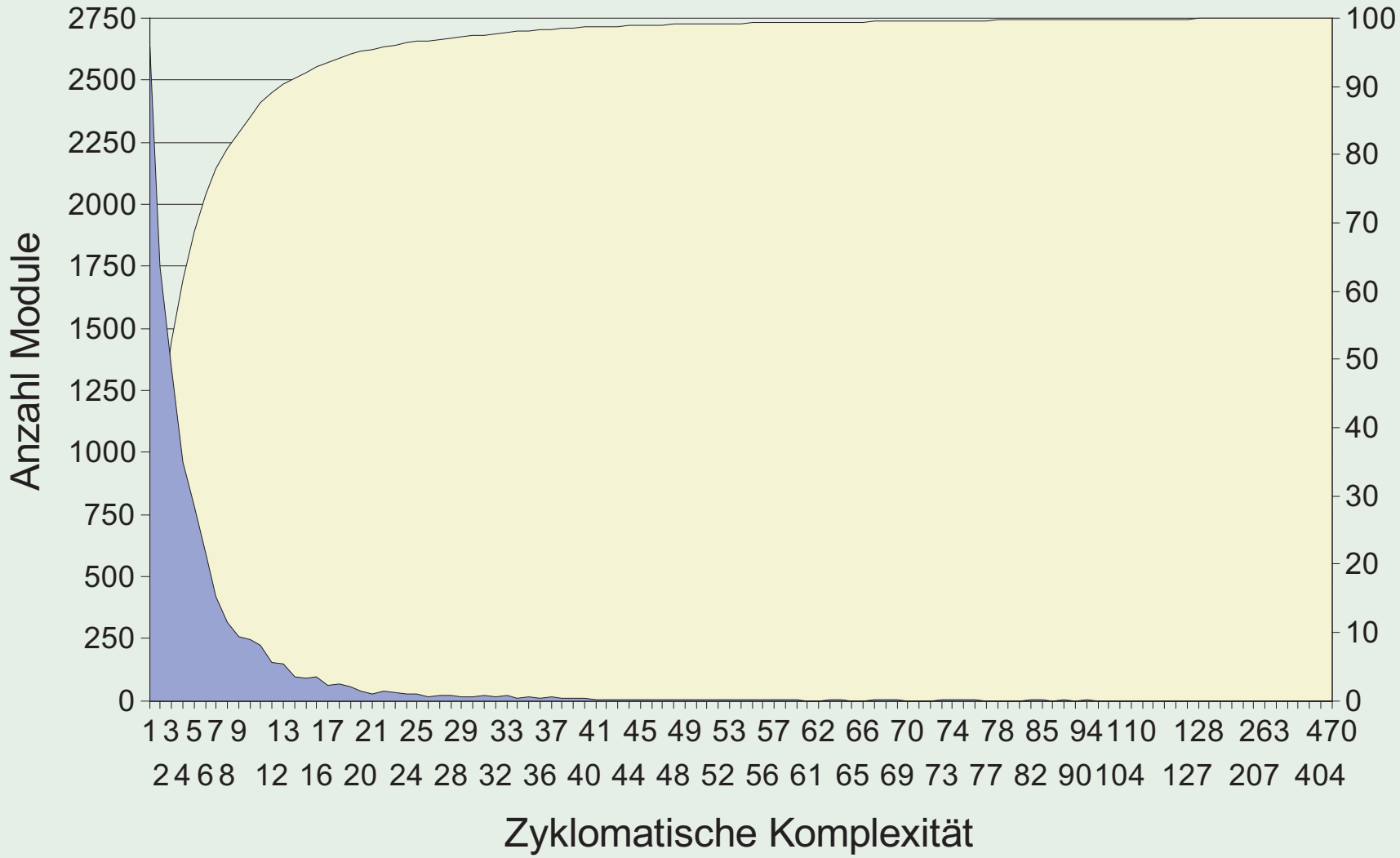
Datensatz

- Insgesamt 10878 Module
- Median von $V(g)$ beträgt 3
- Mittelwert von $V(g)$ beträgt $6,35$

CMU - SEI / Liggesmeyer [CM12005][LIG2002]

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)

JM1


Anzahl Module
2750
2500
2250
2000
1750
1500
1250
1000
750
500
250
0
100
90
80
70
60
50
40
30
20
10
0
1 3 5 7 9 13 17 21 25 29 33 37 41 45 49 53 57 62 66 70 74 78 85 94 110 128 263 470
2 4 6 8 12 16 20 24 28 32 36 40 44 48 52 56 61 65 69 73 77 82 90 104 127 207 404
Zyklomatische Komplexität

Statistik

Gesamtanteil	$V(g)$	Risiko	Anzahl Module (absolut)	$V(g)$
$85,0\ \%$	$\leq\ 10$	niedrig	1631	$>\ 10$
$95,0\ \%$	$\leq\ 20$	mittel	543	$>\ 20$
$97,0\ \%$	$\leq\ 30$	hoch	326	$>\ 30$
$98,6\ \%$	$\leq\ 40$		152	$>\ 40$
$99,1\ \%$	$\leq\ 50$		97	$>\ 50$
$99,9\ \%$	≤ 133	fatal	10	> 133
$100,0\ \%$	≤ 470		10878	$>\ 0$

Fazit JM1

- Ab einer bestimmten Komplexität müssen Module überprüft werden.
  - Schwellenwert nicht allgemein gültig !!
- Metrik für C-Code gut geeignet
- Zu wenig Informationen für Schlußfolgerungen
  - Unklar bleiben Extremwerte (z.B. $V(g) > 300$)
- Potentiell guter bzw. lesbarer Code, da 95% über eine mittlere oder geringere Risikostufe verfügen

Ergänzende Metriken

Complexity Measurement	**Primary Measure of**
Halstead Complexity Measures	Algorithmic complexity, measured by counting operators and operands
Henry and Kafura metrics	Coupling between modules (parameters, global variables, calls)
Bowles metrics	Module and system complexity; coupling via parameters and global variables
Troy and Zweben metrics	Modularity or coupling; complexity of structure (maximum depth of structure chart); calls-to and called-by
Ligier metrics	Modularity of the structure chart

Weitere Vorgehensweise [CM22005]

Maß	Beginn		Refactoring		Erweiterung	
	Funktion	Modul	Funktion	Modul	Funktion	Modul
MI	6,47	33,55	39,93	70,13	37,62	69,60
H. Effort	2.216.499	2.233.072	182.216	480.261	201.429	499.474
McCabe	45	49	18	64	21	67
LOC	622	663	196	732	212	748

Erläuterungen

- MI (=Maintainability Index) : Maß für die Wartbarkeit des Systems
- Halstead Effort schätzt den Aufwand zur Implementierung des Moduls

OO-Metric Suite

Warum spezielle OO-Metriken ?

- Objektorientiertes Design und Programmierung ist weit verbreitet und findet in vielen Projekten Anwendung
- Objektorientierte Software-Entwicklung unterscheidet sich von prozeduraler Software-Entwicklung, weshalb prozedurale Metriken nicht ausreichend sind
- Folgende objektorientiere Strukturen erfordern speziell Metriken:
  - Lokalisation
  - Kapselung
  - Information Hiding
  - Vererbung
  - Abstraktion
- Innerhalb einzelner Methoden einer Klasse sind konventionelle Metriken einsetzbar

Konventionelle Metriken für die objektorientierte Software-Entwicklung

- **L**ines **o**f **C**ode (LOC)
  - Anzahl an Programmzeilen, älteste und bekannteste Metrik
- **M**cCabe's **c**yclomatic **c**omplexity (MCC)
  - Metrik wird anhand eines Flußdiagramms berechnet

Lines of Code (LOC)

- Erfasst die Anzahl an Programmzeilen
- Grundsätzlich geeignet für objektorientierte Software

Voraussetzung und Einschränkung bei Anwendung auf objektorientierte Software

- Die Programmzeilen, welche die objektorientierten Strukturen definieren - z.B. Klassendefinitionen - dürfen nicht mitgezählt werden
- Ergebnisse der LOC-Metrik berücksichtigen nicht die objektorientierte Struktur einer Software. Gemessen wird praktisch nur die Quantität
- Auf Methodenebene ist LOC ohne Einschränkung einsetzbar

McCabe's cyclomatic complexity (MCC)

- Metrik geht nicht vom Source-Code aus, sondern analysiert das Flußdiagramm einer Software
- Grundsätzlich geeignet für objektorientierte Software

Einschränkung

- Die Komplexität eines Programms wird berücksichtigt. Nicht jedoch, dass in einem Knoten beliebig viele Programmzeilen enthalten sein können
- Die objektorientierte Beschaffenheit wird nicht betrachtet

Objektorientierte Designmetriken

Allgemein

- Dienen der Messung der Qualität des Designs
- Beurteilt werden Lesbarkeit und Wartbarkeit des Quellcodes
- Betrachtet werden die Klassen und ihre Beziehungen untereinander
- Objektorientierte Designmetriken können in verschiedene Bezugsebenen unterteilt werden
  - Metriken auf Methodenebene
  - Metriken auf Klassenebene
  - Metriken hinsichtlich Vererbungshierarchien
  - Metriken hinsichtlich Aggregationshierarchie

OO-Metric Suite von Chidamber und Kemerer [SRC1994]

Die OO-Metric Suite besteht aus sechs Metriken

- **W**eighted **M**ethod Per **C**lass (WMC)
- **D**epth of **I**nheritance **T**ree (DIT)
- **N**umber **o**f **C**hildren (NOC)
- **C**oupling **b**etween **o**bject classes (CBO)
- **R**esponse **f**or a **C**lass (RFC)
- **L**ack of **Co**hesion in **M**ethods (LCOM)

Anmerkung

- Die aufgeführten OO-Metriken messen die Komplexität im Design der Klassen. Mögliches dynamisches Verhalten eines Systems wird nicht gemessen
- Es handelt sich somit um statische Metriken, welche vor Programmausführung gemessen werden können

NASA Metrics Data Program [NAS2006]

Einsatz der sechs OO-Metriken anhand des KC1 Projekts der NASA [NAS2006]

- KC1 - This is a single CSCI within a large ground system. It is made up of 43 KSLOC of C++ code. Error data for this code has been collected since the beginning of the project, but that data can only be associated to the module (classes) back to five years.

Metrik 1: **W**eighted **M**ethod Per **C**lass (WMC)

Berechnung [SRC1994]

$$WMC(C) = \sum_{i=1}^{n} c_i$$

Definition

- Betrachtet man die Methoden $M_1 ... M_n$ der Klasse C, dann wird die Komplexität der Methoden durch $c_1 ... c_n$ beschrieben. Somit berechnet $WMC(C)$ die Summe der jeweiligen Komplexität der einzelnen Methoden der Klasse C
- Meist werden lokal definierte Methoden einschließlich Redefinitionen gezählt

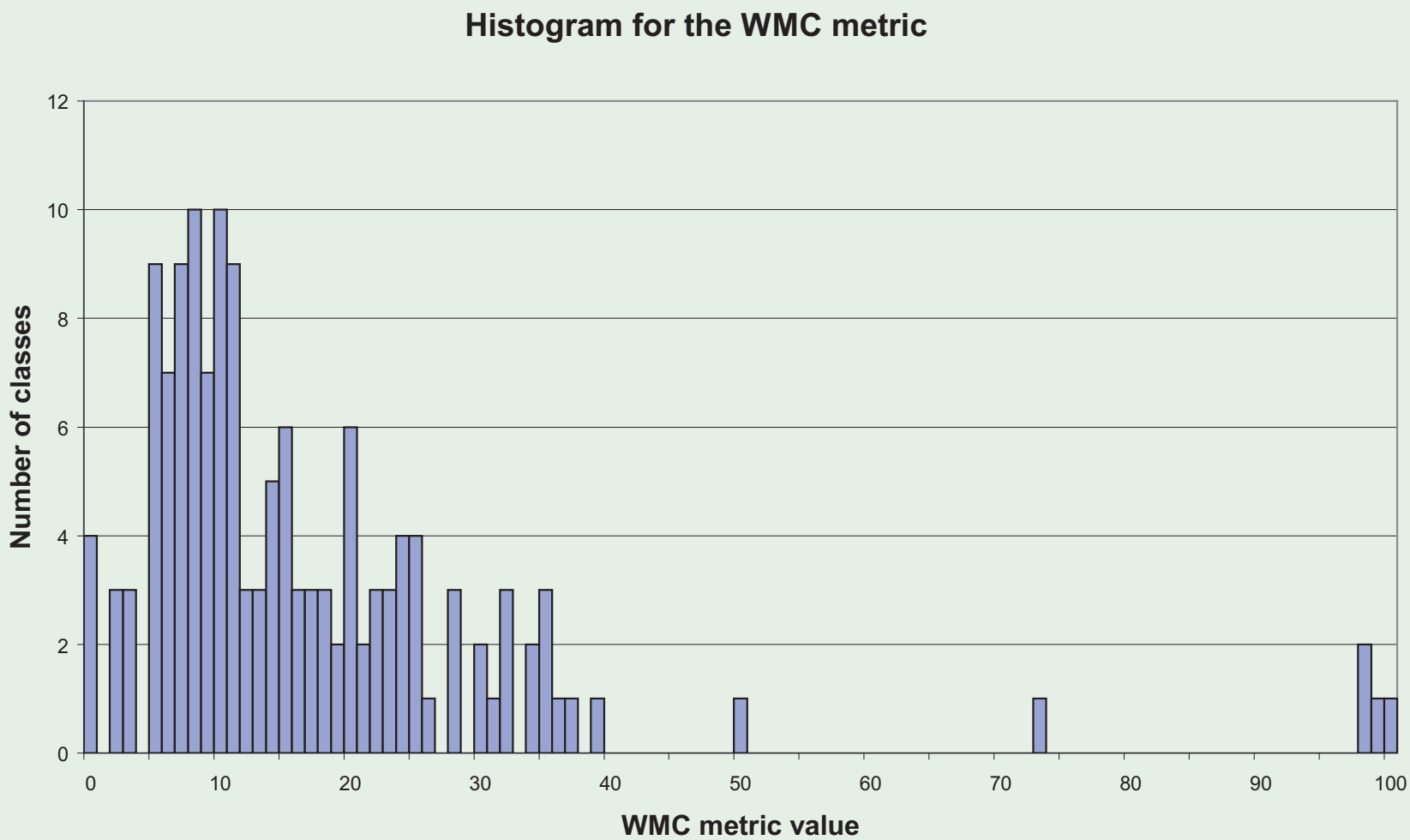
Definition der Komplexität

- Metrik zur Bestimmung der Komplexität der Methoden steht offen
- Mögliche einsetzbare Metriken sind:
  - **L**ines **o**f **C**ode (LOC)
  - **M**cCabe's **c**yclomatic **c**omplexity (MCC)
  - Bewertung einer Methode mit einer Standardkomplexität von 1

Interpretation

- Anzahl und Komplexität der Methoden bestimmen den Aufwand der Entwicklung und Wartung einer Klasse
- Umso größer die Anzahl an Methoden innerhalb einer Klasse, desto größer ist der Einfluss auf abgeleitete Klassen welche die Methoden erben
- Fehlerwahrscheinlichkeit nimmt mit der Anzahl an Methoden zu
- Klassen die viele Methoden beherbergen sind meist sehr applikationsspezifisch und somit tendenziell schlechter wiederverwendbar

KC1 - Metrik 1: **W**eighted **M**ethod Per **C**lass (WMC)



Statistics for the WMC Metric

NASA Project	Metric	Median	Max	Min
*KC*1	*WMC*	12	100	0

Interpretation der Daten

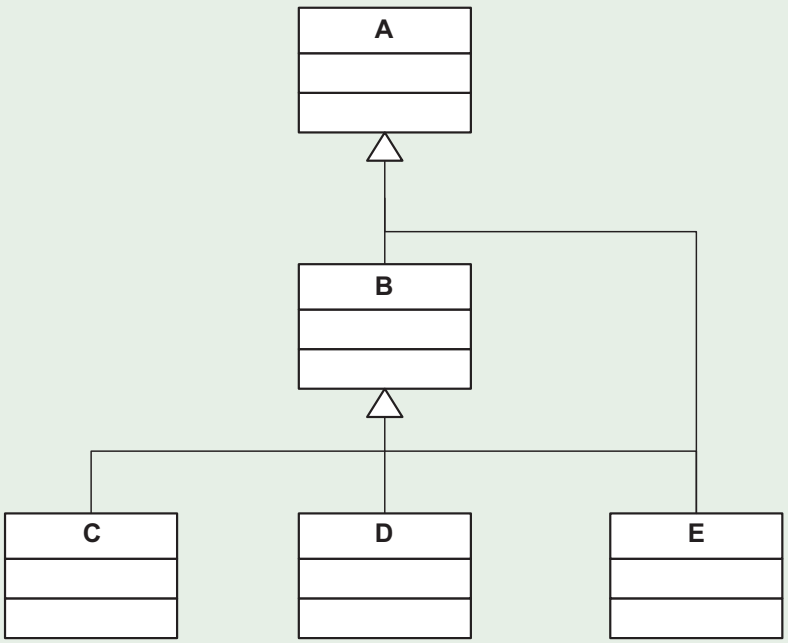
- Die Klasse mit dem *WMC* Wert von 99 hat keine abgeleitete Klassen und steht an oberster Stelle im Ableitungsbaum. Somit finden die Klassenmethoden keine Wiederverwendung innerhalb dieser Applikation
- Die Klasse mit dem *WMC* Wert von 26 hat 5 direkte von ihr abgeleitete Klassen und steht an 4. Stelle von 7 Ebenen im Ableitungsbaum.
  $\Rightarrow$ Hoher Wiederverwendungswert, Methoden sind weit im System verbreitet
  $\Rightarrow$ Methoden sollten genau getestet werden

Metrik 2: **D**epth of **I**nheritance **T**ree (DIT)

Berechnung [SRC1994]

- $DIT(C)$ ist die Länge des maximalen Weges in der Klassenhierarchie von der Wurzel bis zur Klasse C

Beispiel *DIT* [SRC1994]



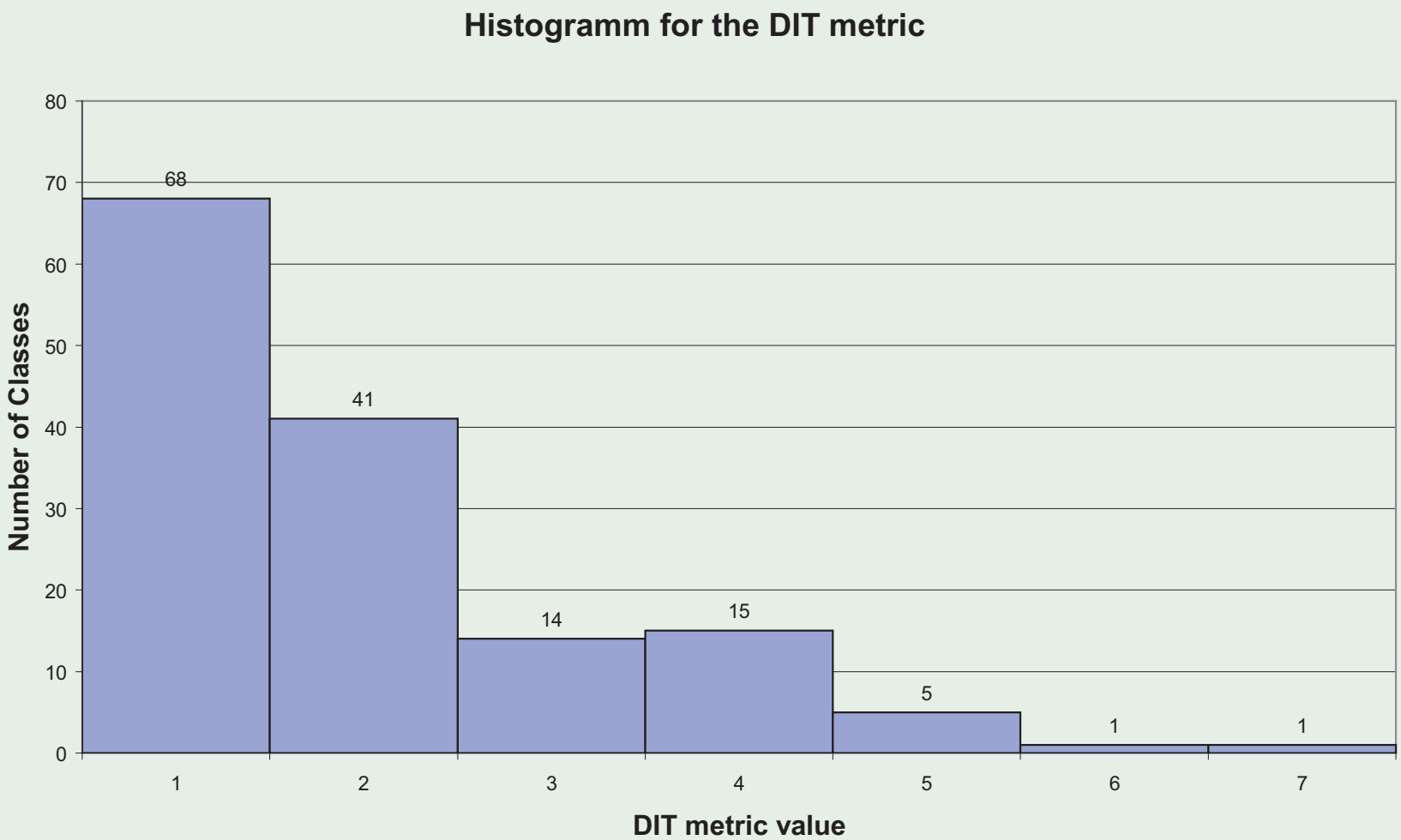
Beispiel

- $DIT(A) = 0$
- $DIT(B) = 1$
- $DIT(C) = 2$
- $DIT(D) = 2$
- $DIT(E) = 2$

Interpretation

- *DIT* ist ein Maß für die Anzahl an Vorgängern, die Einfluss auf eine Klasse nehmen
- Metrik liefert Aussagen über Komplexität und Wiederverwendbarkeit
- Klassen mit vielen Oberklassen sind komplexer und schwerer zu handhaben, da sie mehrere Methoden erben
- Große Vererbungshierarchien sind schwer zu entwerfen, da viele Methoden und Klassen beteiligt sind
- Klassen die sich tiefer in der Vererbungshierarchie befinden haben eine höhere Wiederverwendung an ererbten Methoden

KC1 - Metrik 2: **D**epth of **I**nheritance **T**ree (DIT)



Statistics for the DIT Metric

NASA Project	Metric	Median	Max	Min
*KC*1	*DIT*	2	7	1

Interpretation der Daten

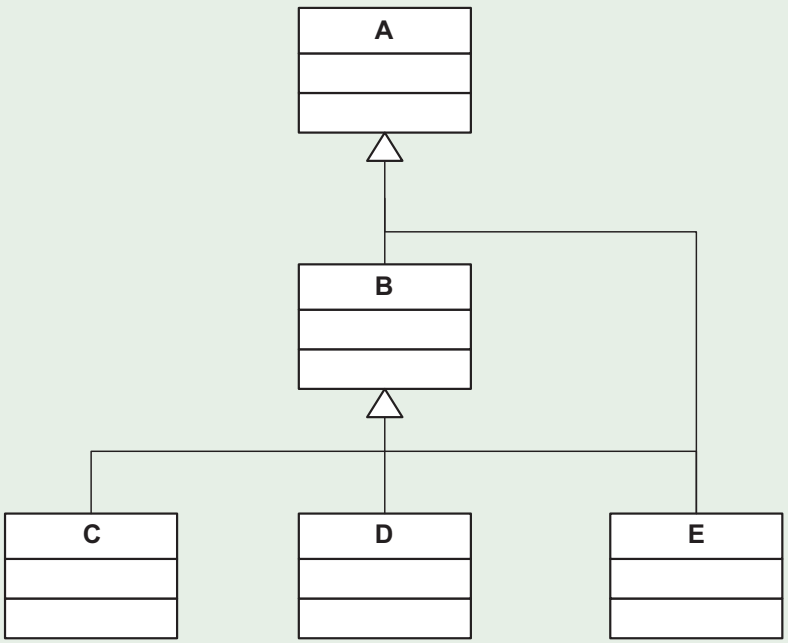
- NASA KC1-Projekt-Studie: Root wird in der Vererbungshierarchie mit *DIT* Value 1 angegeben
- Metrik kann eingesetzt werden um Aussagen über Design Aufteilung zu treffen
  - „top heavy“ $\Rightarrow$ Großteil der Klassen befinden sich nahe der Wurzel in der Vererbungshierarchie
  - „down heavy“ $\Rightarrow$ Großteil der Klassen befinden sich näher am Boden in der Vererbungshierarchie
- Niedriger *DIT* KC1 Median $\Rightarrow$ „top heavy“

Metrik 3: **N**umber **o**f **C**hildren (NOC)

Berechnung [SRC1994]

- $NOC(C)$ beschreibt die Anzahl an direkten Unterklassen der Klasse C

Beispiel *NOC* [SRC1994]



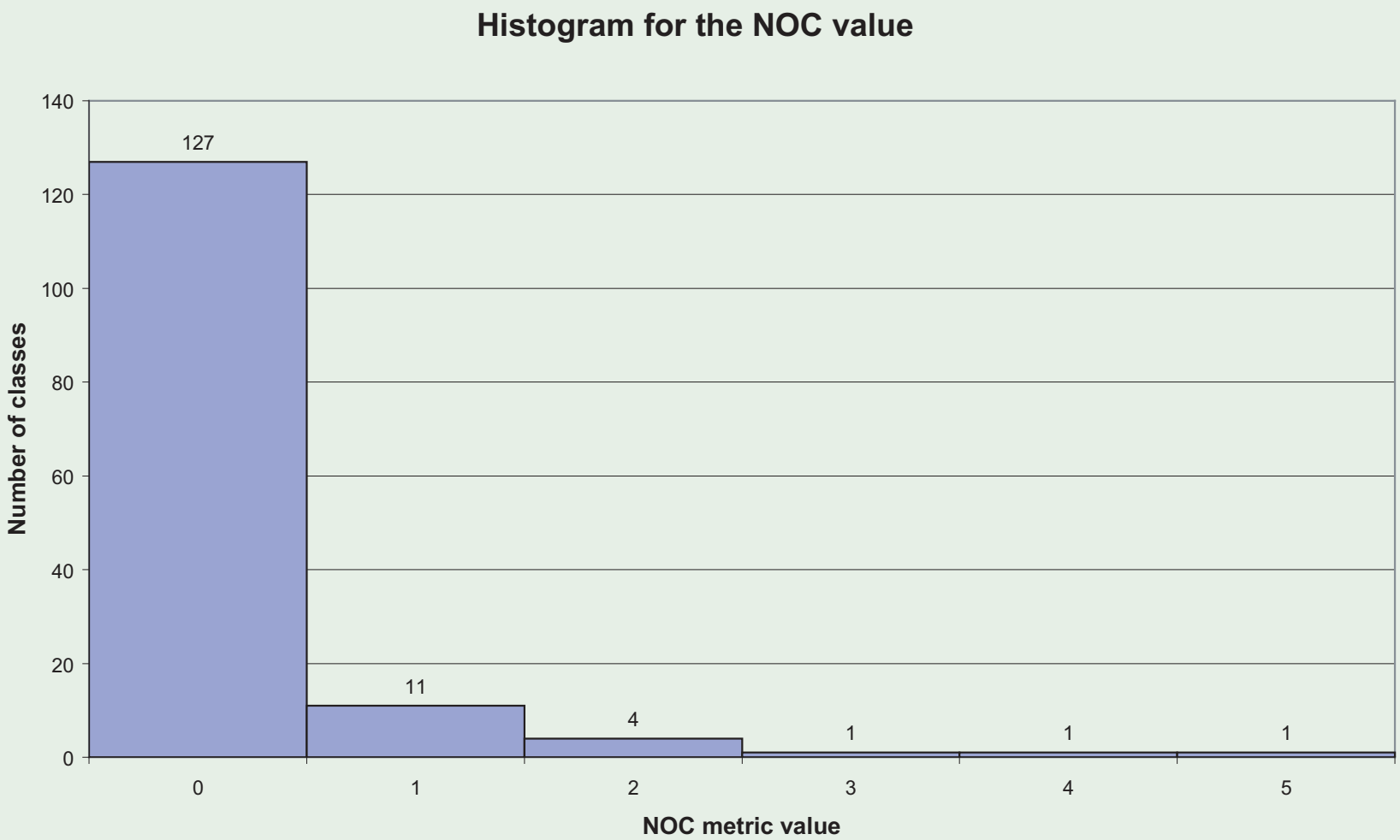
Beispiel

- $NOC(A) = 2$
- $NOC(B) = 3$
- $NOC(C) = 0$
- $NOC(D) = 0$
- $NOC(E) = 0$

Interpretation

- Je größer die Zahl an direkten Nachfolger, desto höherer Wiederverwendungsgrad durch Vererbung liegt vor
- Klassen mit vielen Kindern sind entwicklungsintensiver und sollten evtl. umfangreicheren Tests unterzogen werden
- Klassen mit vielen Kindern sollten dahingehend überprüft werden, ob Kinder fachlich korrekt von der Klasse erben, um fehlerhaftes Anwenden der Vererbung zu verhindern

KC1 - Metrik 3: **N**umber **o**f **C**hildren (NOC)



Statistics for the NOC Metric

NASA Project	Metric	Median	Max	Min
*KC*1	*NOC*	0	5	0

Interpretation der Daten

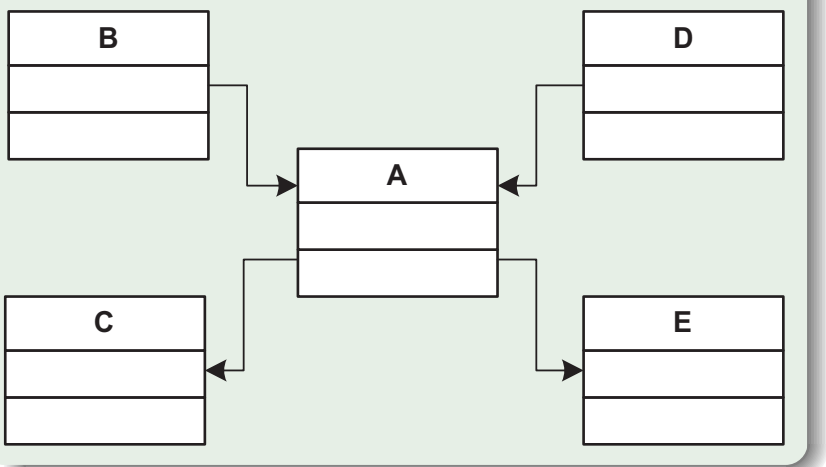
- Großteil der Klassen besitzen keine direkten Nachfolger. Einige wenige Sonderfälle haben mehrere direkte Nachfolger.
  $\Rightarrow$ Mögliche Interpretation:
  - Designer verwenden Methodenvererbungkonzept nicht als zentrale Basis im Klassendesign
  - Schlechte Kommunikation zwischen den verschiedenen Klassendesignern und somit geringe Wiederverwendung von Klassen

Metrik 4: **C**oupling **b**etween **o**bject classes (CBO)

Berechnung [SRC1994]

- $CBO(C)$ Metrik beschreibt die Anzahl an Klassen, mit denen die Klasse C gekoppelt ist
- Eine Klasse A ist an eine Klasse B gekoppelt, wenn Methoden von A Methoden oder Instanzvariablen von B verwenden

Beispiel *CBO* [SRC1994]



Beispiel

- $CBO(A) = 2$

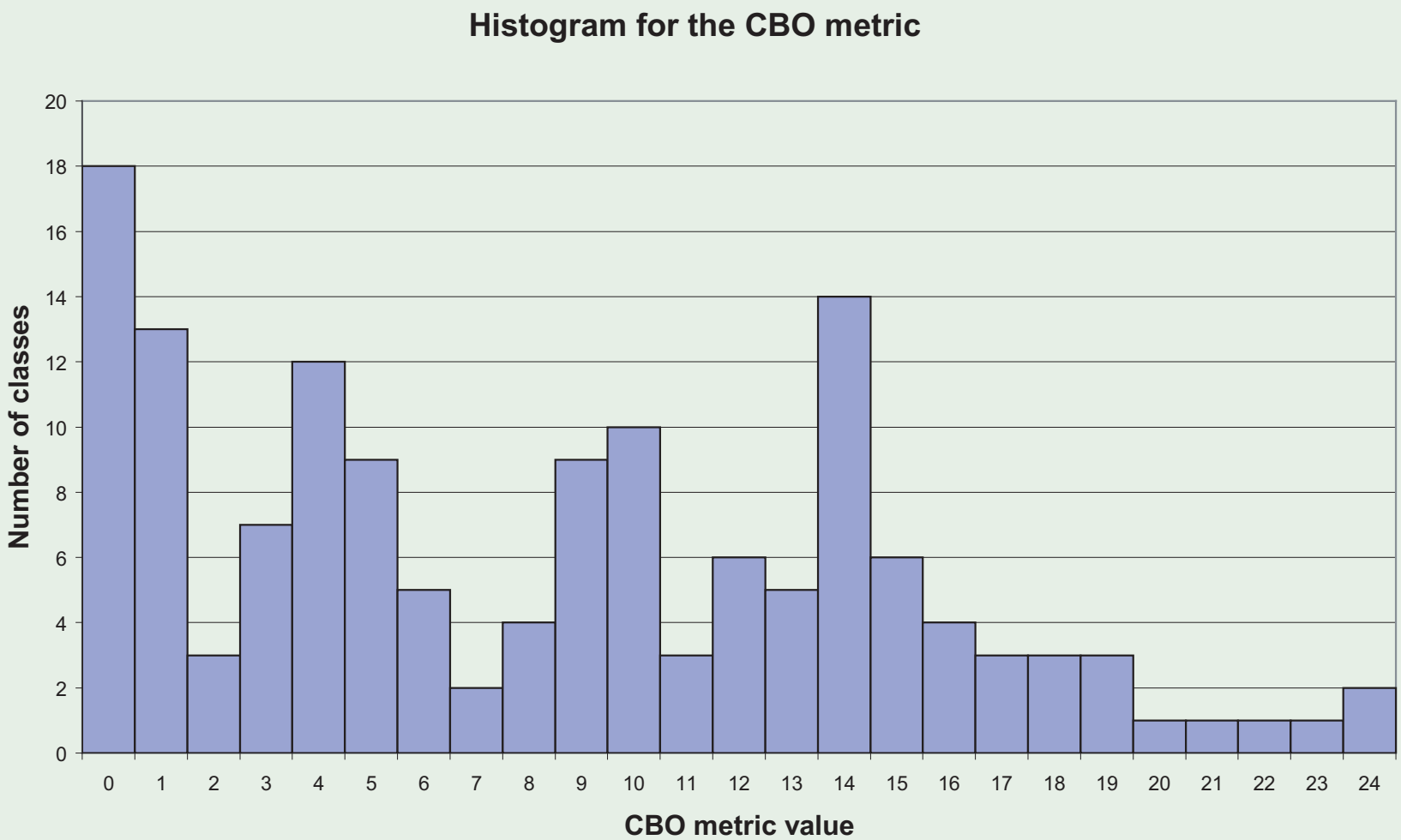
Interpretation

- Exzessive Kopplung zwischen Klassen verhindert Wiederverwendung und widerspricht dem Konzept der Modularisierung
  $\Rightarrow$ Desto unabhäniger eine Klasse ist desto einfacher ist diese in anderes Applikationen wiederverwendbar
- Durchführung von Änderungen wird bei exzessiver Kopplung komplizierter
- Starke Kopplung führt zu einem erhöhten Wartungsaufwand
- Starke Kopplung führt zu zusätzlichen und komplexeren Testfällen

Folgerung

- Kopplung sollte möglichst gering gehalten werden

KC1 - Metrik 4: **C**oupling **b**etween **o**bject classes (CBO)



Statistics for the CBO Metric

NASA Project	Metric	Median	Max	Min
KC1	*CBO*	8	24	0

Interpretation der Daten

- Der hohe Median-Wert suggeriert, daß nur 12 % aller Klassen nicht mit anderen Klassen gekoppelt sind
- Konzept der Modularisierung und Kapselung wurde vernachlässigt
  - Hoher Wartungsaufwand
  - komplexere Testfälle
  - Änderungen schwerer realisierbar
- *CBO* Metrik kann von Projekt Managern eingesetzt werden um die Klassenhierachie auf möglichen Integritätsverlust zu überprüfen
- *CBO* Metrik kann herangezogen werden um Aussagen über ungeeigneten Einsatz von Kopplungen innerhalb bestimmter Stellen im System zu treffen

Metrik 5: **R**esponse **f**or a **C**lass (RFC)

Berechnung [SRC1994]

$$RFC = |RS|$$

$$RS = \{M\} \cup_{all\ i} \{R_i\}$$

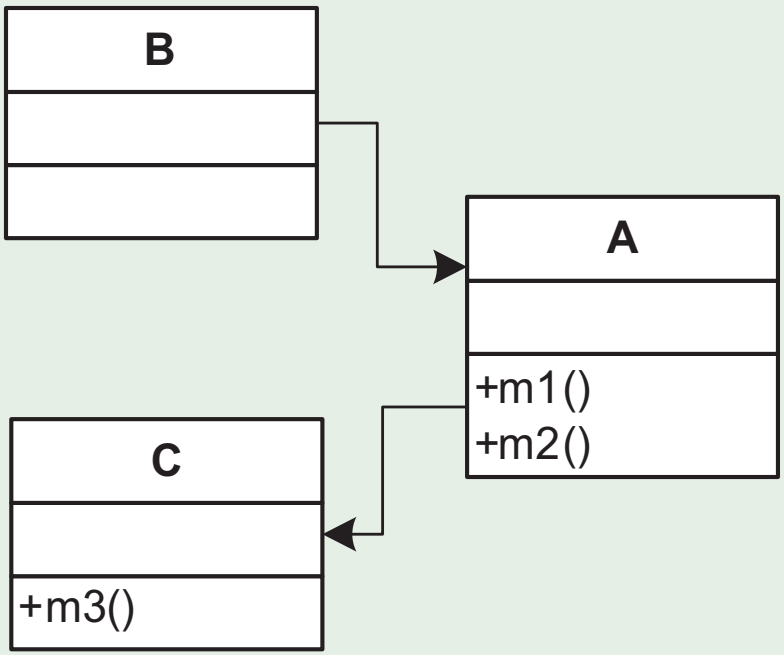
Variablen

- RS: Response-set der Klasse
- M: Anzahl aller Methoden innerhalb der Klasse
- R_i: Anzahl der Methoden die von der Methode i aufgerufen werden
- $RFC(C)$ Anzahl an Methoden, die potentiell ausgeführt werden können, wenn ein Objekt der Klasse C auf eine eingegangene Nachricht reagiert

Berechnung [SRC1994]

- Die Größe der Response-Menge einer Klasse ergibt sich aus der Anzahl der Methoden einer Klasse plus die Anzahl der Methoden anderer Klassen, die von den Methoden der Klasse benutzt werden
- Jede Methode wird nur einmal gezählt

Beispiel *CBO* [SRC1994]



Beispiel

- $RFC(A) = 3$
- $RFC(B) = 0$
- $RFC(C) = 1$

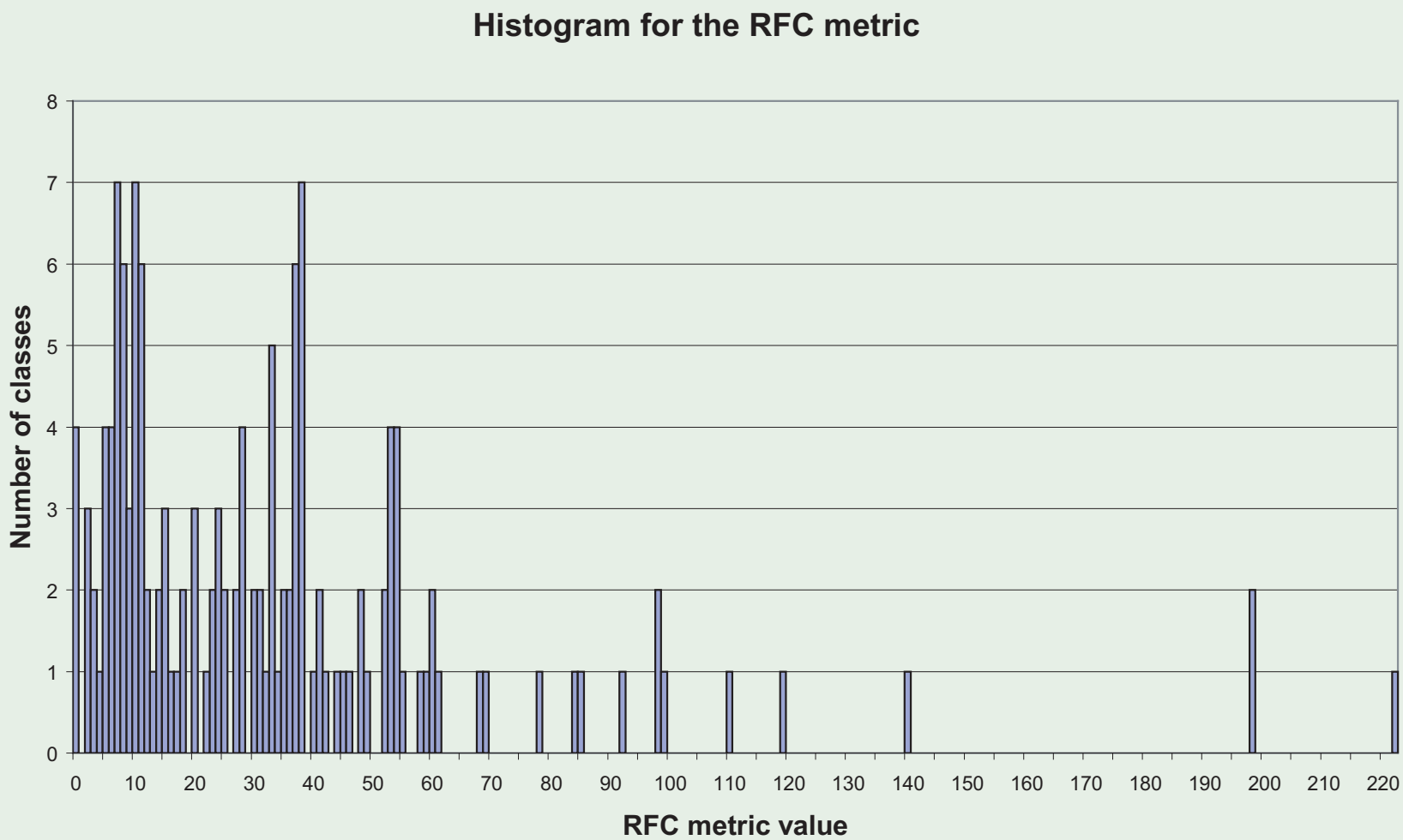
Interpretation

- Je größer der RFC-Wert einer Klasse, desto komplexer und fehleranfälliger ist eine Klasse
  - Erhöhter Test und Debugging Aufwand

Folgerung

- Response set sollte möglichst gering gehalten werden

KC1 - Metrik 5: **R**esponse **f**or a **C**lass (RFC)



Statistics for the RFC Metric

NASA Project	Metric	Median	Max	Min
KC1	*RFC*	28	222	0

Interpretation der Daten

- Großteil der Klassen kann nur eine geringe Anzahl an Methoden aufrufen, wobei einige wenige Ausnahmen sehr exzessiv Methoden aufrufen können
- Eine geringe Anzahl an Klassen ist für eine große Anzahl an Methoden die in dem System ausgeführt werden verantwortlich, entweder weil diese Klassen viele Methoden enthalten oder weil von ihnen viele Methoden aufgerufen werden
- Unter Einsatz von Klassen mit hohem RFC Werten kann während des System-Tests eine hohe Testabdeckung erreicht werden
- Metrik stellt Managern und Designern eine Basis zur Verfügung um Design der Klassenhierarchie zu untersuchen

Metrik 6: **L**ack of **Co**hesion in **M**ethods (LCOM)

Berechnung [SRC1994]

$$LCOM(C) = \begin{cases} |P| - |Q| & \text{, wenn } |P| > |Q| \\ 0 & \text{, sonst} \end{cases}$$

Variablen

- Betrachtet man eine Klasse C mit n Methoden $M_1 ... M_n$ wobei $\{I_j\}$ die Menge an Instanzvariablen beschreibt, die von der Methode M_i benutzt werden
- Betrachtet man alle n Mengen dann ist $P = \{(I_i, I_j) \,|\, I_i \cap I_j = 0\}$ und $Q = \{(I_i, I_j) \,|\, I_i \cap I_j \neq 0\}$
- Falls alle n Mengen $I_1 ... I_n$ gleich 0 sind, dann ist $P = 0$

Berechnung [SRC1994]

- $LCOM(C)$ = Anzahl der Paare an Methoden der Klasse C ohne gemeinsame Instanzvariablen minus der Anzahl der Paare von Methoden der Klasse C mit gemeinsamen Instanzvariablen.

Beispiel $LCOM$ [SRC1994]

Klasse C	
Methode M1(Var. a,b)	**Methode M2(Var. a,c)**
Methode M3(Var. c,d)	**Methode M4(Var. x,y)**

- 4 Paare ohne gemeinsame Instanzvariablen: M1-M3, M1-M4, M2-M4, M3-M4
- 2 Paare mit gemeinsamen Instanzvariablen: M1-M2, M2-M3
- $LCOM(C) = 4 - 2 = 2$

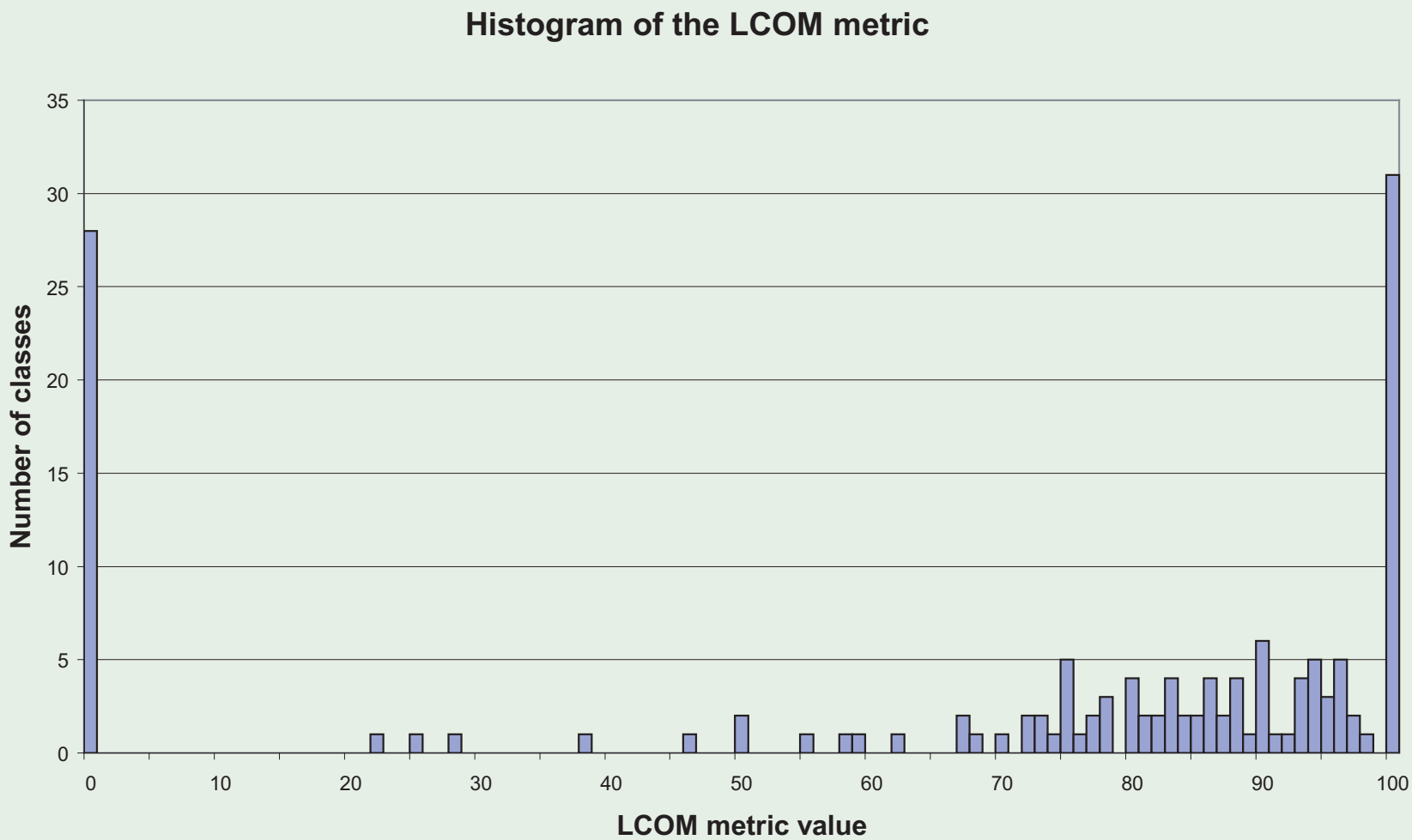
Interpretation

- Kohäsion innerhalb einer Klasse ist wünschenswert
  $\Rightarrow$ Indiz für gute Kapselung
- Geringe oder nicht vorhandene Kohäsion deutet auf Fehler im Design hin
  $\Rightarrow$ Klassen sollten möglicherweise aufgeteilt werden
- Eine niedrige Kohäsion erhöht die Komplexität und damit auch die Fehleranfälligkeit der Software

Folgerung

- LCOM sollte möglichst gering gehalten werden

KC1 - Metrik 6: **L**ack of **Co**hesion in **M**ethods (LCOM)



Statistics for the LCOM Metric

NASA Project	Metric	Median	Max	Min
KC1	*LCOM*	84%	100%	0%

Interpretation der Daten

- Aufteilung der Daten in %
  - Niedriger % Wert $\Rightarrow$ Kohäsion innerhalb der Klasse
  - Hoher % Wert $\Rightarrow$ Niedrige Kohäsion innerhalb der Klasse (Klasse sollte aufgeteilt werden)
- Median von 84% signalisert, dass Großteil der Methoden innerhalb der Klassen eine niedrige Kohäsion besitzen $\Rightarrow$ Einige Klassen implementieren viele verschiedene Aufgaben. Aufgaben sollten auf weitere Klassen aufgeteilt werden
- Metrik stellt Managern und Designern eine Möglichkeit zur Verfügung, festzustellen ob Kohäsionsrichtlinien im Design eingeflossen sind. Falls notwendig können Änderungen in frühen Phasen des Designzyklus durchgeführt werden

Mapping of metrics to OOD steps

Metric	Identification	Semantics	Relationships
WMC	X	X	
DIT	X		
NOC	X		
RFC		X	X
CBO			X
LCOM		X	

Zusammenfassung 1/2

- **Identification of classes:**
  - WMC, DIT und NOC wird im Design bei der Identifikation der Klassen eingesetzt
    - WMC zur Messung der Komplexität der Klasse
    - DIT und NOC zur Feststellung der Klassenhierarchie

Zusammenfassung 2/2

- **Semantics of classes:**
  - WMC, RFC erfassen wie sich Objekte einer Klasse auf Nachrichten verhalten können
    - Hohe WMC und RFC Werte zeugen von vielen möglichen Antwortquellen, da viele Methoden ausgeführt werden können
  - LCOM Metrik bezieht sich auf die Daten- und Methodenkapselung innerhalb einer Klassendefinition und gibt Auskunft über die Kohäsion einer Klasse
- **Relationships between classes:**
  - RFC und CBO erfassen den Kommunikationsumfang zwischen Klassen indem die Klassen- und Methodenbeziehungen gezählt werden

Gütekriterien für Metriken (1/2)

- **Objektivität** ⇒ Kein subjektiver Einfluss durch Prüfer möglich
- **Zuverlässigkeit** ⇒ Wiederholung liefert gleiche Ergebnisse
- **Nützlichkeitsfaktor** ⇒ Praktische Bedürfnisse werden erfüllt
- **Normierung** ⇒ Es gibt eine Skala für die Messergebnisse
- **Vergleichbarkeit** ⇒ Mit anderen Maßen vergleichbar
- **Ökonomie** ⇒ Messung mit geringen Kosten durchführbar
- **Messtauglichkeit** ⇒ Messergebnisse erlauben Rückschluss auf Ausprägung der Qualitätseigenschaft
- **Skalierbarkeit** ⇒ Einsetzbar sowohl für große, als auch für kleine Projekte

Gütekriterien für Metriken (2/2)

- **Anpassbarkeit** ⇒ Je nach Zielsetzung werden unterschiedliche Abstraktionen des Quelltexts untersucht
- **Unschärfe** ⇒ Tatsächliche Implementierung nicht notwendig
- **Effizienz und Effektivität** ⇒ Erkennen von Schwachstellen bei guter Wahl der Metriken

Vorteile von OO-Metriken

- Ansatzpunkt für präventive Wartung
- Softwareentwicklung wird vorhersagbarer
- Test- und Wartungsaufwand beurteilen
- Ergänzung der Programmierrichtlinien
- Schwachstellen identifizieren
- Kundenanforderungen überprüfbar machen
- Erzieherischer Effekt auf den Programmierer

Schwachstellen von OO-Metriken

- Gesamtkomplexität eines Systems kann schwerlich durch einen Wert charakterisiert werden
- In der Regel werden nur einfache Sachverhalte gezählt (kein empirischer Beweis für tatsächliche Qualität)
- Viele objektorientierte Metriken meßtechnisch unzureichend definiert
- Objektorientierte Analyse und objektorientierter Entwurf werden noch nicht behandelt

Zusammenfassung

- Metriken sind einfach zu erfassen, aber schwierig zu interpretieren.
- Die zyklomatische Komplexität ermöglicht eine schnelle Identifikation potentieller Schwachstellen.
- Objektorientierte Metriken sind auf objektorientierte Strukturen ausgerichtet und helfen bei der Qualitätssicherung
- Analyse und Entwurf werden von den OO-Metriken nicht abgedeckt.

Open Source Tools

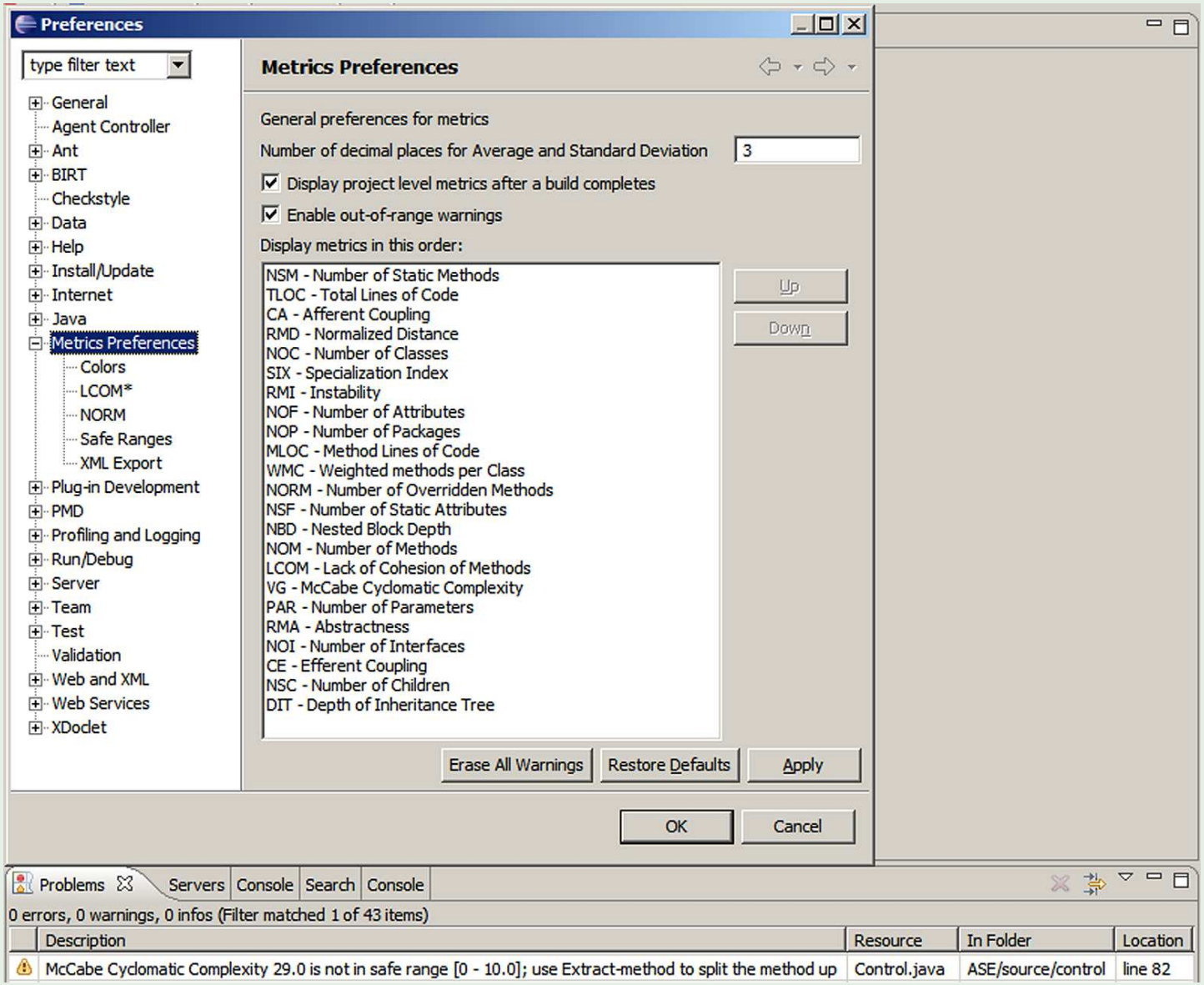
Eclipse

- Metrics *http://www.teaminabox.co.uk/downloads/metrics/*
- Metrics *http://metrics.sourceforge.net/*

Features

- Ausgabe / Export in Tabellenform
- Marker im Quelltext bei Überschreitung von Schwellenwerten

Eclipse Metrics Plugin



Kommerzielle Tools

Verschiedene Sprachen

- McCabe IQ *http://www.mccabe.com*
- Weitere unter *http://www.laatuk.com/tools/metric_tools.html*

Literaturangaben I

[CM12005] Carnegie Mellon University - Software Engineering Institute
Cyclomatic Complexity
http://www.sei.cmu.edu/str/descriptions/cyclomatic_body.html

[CM22005] Carnegie Mellon University - Software Engineering Institute
Maintainability Index Technique for Measuring Program Maintainability
http://www.sei.cmu.edu/str/descriptions/mitmpm.html

[KP2006] Kantelinen, Pasi
Software metrics tools
http://www.laatuk.com/tools/metric_tools.html

[METR2005] Fink, Miriam; Hampp, Tilmann
Eine Untersuchung zum Metrikeinsatz in der Industrie
http://www.iste.uni-stuttgart.de/se/publications/download/Metrikon05.pdf

Literaturangaben II

[NAS2006] Callis, Pat
NASA Metrics Data Program
http://mdp.ivv.nasa.gov

[SRC1994] Chidamber, Shyam R. and Kemerer, Chris F.
A metrics Suite for Object Oriented Design
IEEE Transactions on software engineering, Volume 20, Number 6, June 1994

[TJM1976] McCabe, Thomas J.
A complexity measure
IEEE Transactions on software engineering, Volume SE-2, Number 4, Dezember 1976

[ZH1999] Zuse, Horst Dr.
Validation of Measures and Prediction Models
http://paginaspersonales.deusto.es/cortazar/doctorado/articulos/zuse99.pdf

Literaturangaben III

[ZH11995] Zuse, Horst Dr.
History of Software Measurement
http://irb.cs.tu-berlin.de/~zuse/metrics/3-hist.html

[ZH21995] Zuse, Horst Dr.
Is Measurement a Simple Thing?
http://irb.cs.tu-berlin.de/~zuse/metrics/lecture01.html

[LIG2002] Liggesmeyer, Peter Prof. Dr.
Software-Qualität
Spektrum-Verlag 2002
ISBN 3-8274-1118-1