Qualitätssicherung von Software

Prof. Dr. Holger Schlingloff

Humboldt-Universität zu Berlin
und
Fraunhofer FIRST

Wo stehen wir?

1. Einleitung, Begriffe, Software-Qualitätskriterien
2. manuelle und automatisierte Testverfahren
3. Verifikation und Validierung, Modellprüfung
4. statische und dynamische Analysetechniken
5. Softwarebewertung, Softwaremetriken
6. Codereview- und andere Inspektionsverfahren
7. Zuverlässigkeitstheorie, Fehlerbaumanalyse
8. Qualitätsstandards, Qualitätsmanagement, organisatorische Maßnahmen

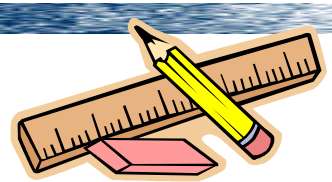
Softwarebewertung

- Qualität = Übereinstimmung mit den Anforderungen
- Validation und Verifikation: Nachweis der Korrektheit (funktionale Anforderungen)
- nicht-funktionale Anforderungen: Verständlichkeit, Dokumentiertheit, Wartbarkeit, Portierbarkeit, Erweiterbarkeit, Modularität, Programmierstil, ...
- Methoden: Messen, Visualisieren, Schätzen, Begutachten

Softwaremetrik

- Wichtigste Methode zur Bewertung von Software
- Messen von bestimmten Größen im Quelltext und Vergleich mit Sollvorgaben
- unterschiedliche Positionen
  - „You can‘t control what you can‘t measure“
  - "Not everything that counts
    is countable; and not everything that is countable
    counts„
- Frage: Was kann/soll gemessen werden?

Definitionen für Software-Metriken



- Sommerville

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

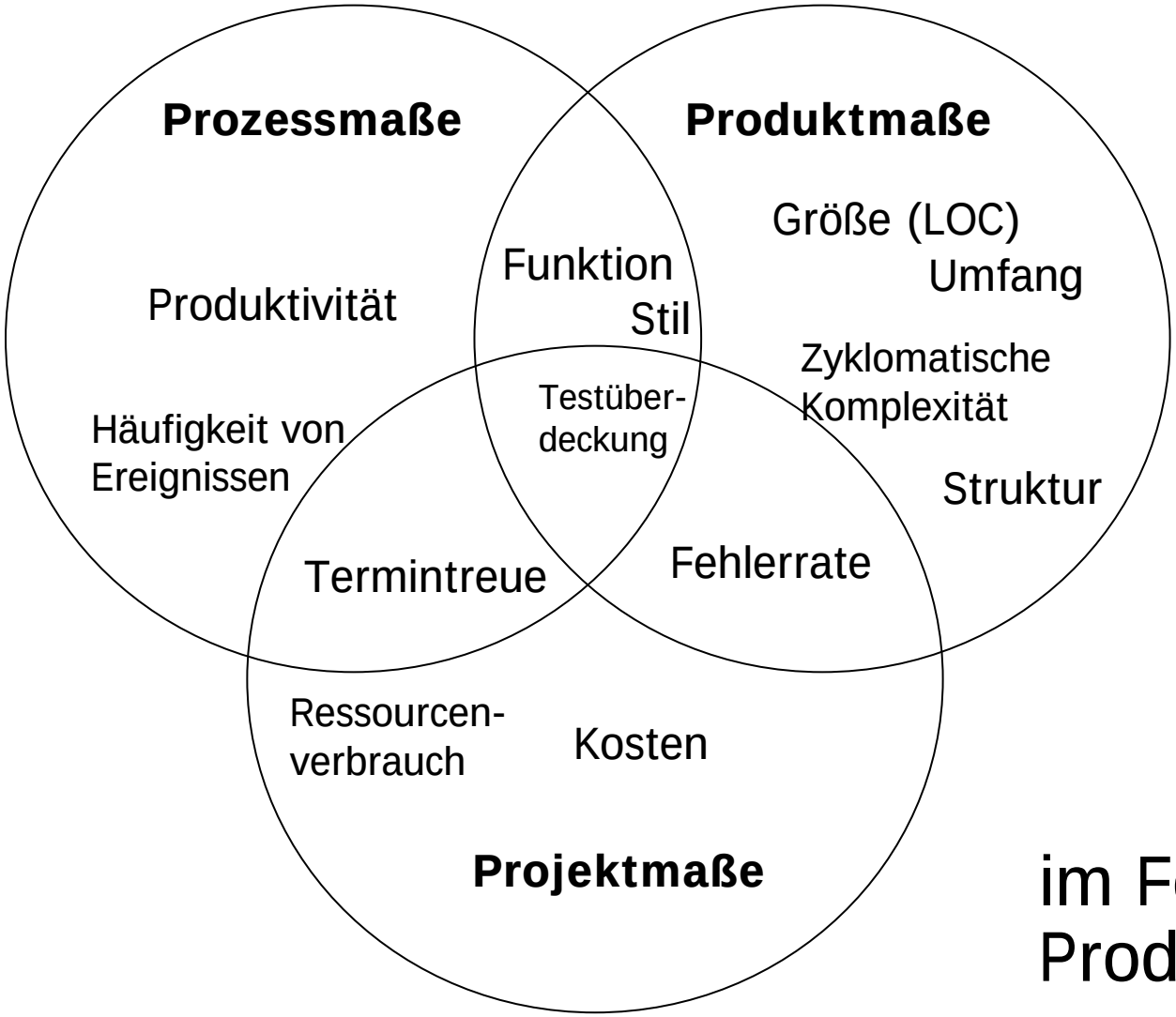
- Liggesmeyer

  „Größe zur Messung einer bestimmten Eigenschaft eines Programms oder Moduls."

- 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.“

Typen von Maßen



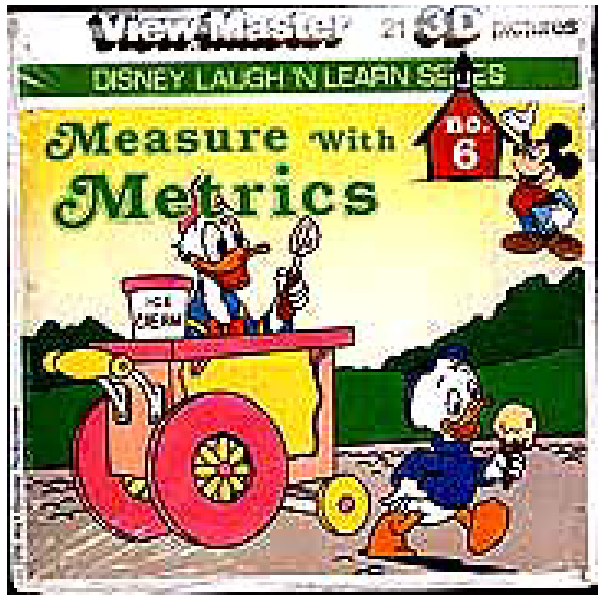
im Folgenden:
Produktmaße

Forderungen an Maße (Gütekriterien)

- Eignung, Gültigkeit
  - Korrelation Maßeinheit / Eigenschaft
- Einfachheit, Ökonomie
  - Aufwand für Messung
  - Aufwand für Interpretation
- Stetigkeit
  - keine Sprünge bei kleinen Änderungen
  - Stabilität gegenüber Manipulation / Tuning
- Reproduzierbarkeit
  - keine Beeinflussung durch Messprozess
  - keine subjektiven Einflüsse
- Rechtzeitigkeit, Analysierbarkeit
  - Messung für Verbesserungen verwendbar

Negativbeispiel

- Messung Programmkomplexität in LOC
  - Kommentare, Leerzeilen?
  - Lange Zeilen? Anweisungen?
  - Präprozessor-Anweisungen?
  - Makros? expandierter Code?
  - Strukturiertheit, Vererbung?



  - Programmierstil? Sprache? Bibliotheken?
  - Korrelation oder Anti-Korrelation?

Alternativen?

Anzahl der

- Zeichen,
- Module, Klassen,
- Variablen,
- Identifier,
- Funktionen,
- Anweisungen,
- Benutzungselemente,
- Schnittstellen,
- ...

„Es gibt derzeit keine allgemein akzeptierten Meßmethoden für die Softwarequalität.“
(DIN, ISO /ISO 9000-3: 1992/)

mehr als tausend Qualitäts- und Produktivitätsmaße in der Literatur!

Metriken für die Software-Qualität

- Ausfälle je Zeiteinheit unter realen Betriebsbedingungen
- Anrufe von Kunden beim Kundendienst je Monat
- Anzahl der in den ersten drei Monaten nach Inbetriebnahme gefundenen Fehler / Entwicklungsaufwand in Personen-Monaten
- Verhältnis zwischen
  - *gefundenen und behobenen Fehlern*
  - *Anzahl der gefundenen Fehler und Programmgröße in Kloc*
- tatsächlicher Testüberdeckungsgrad

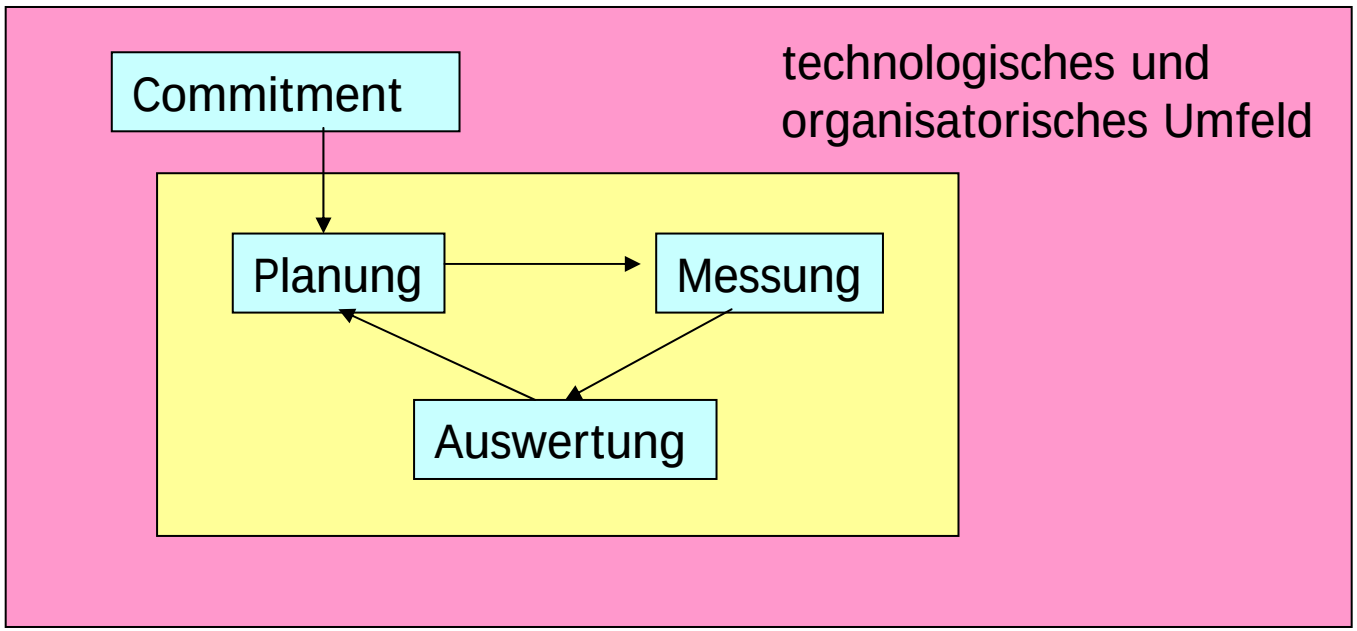
Pause

Sind Metriken überhaupt geeignet?

- Annahmen
  - Die Metrik misst bestimmte Werte der Software.
  - Diese Werte werden durch bestimmte Eigenschaften der Software beeinflusst. Die Beziehung zwischen Metrik und Eigenschaft ist stabil und einigermaßen verstanden.
  - Ziel ist es, die untersuchten Eigenschaften der Software nach Möglichkeit zu verbessern.
- Dann
  - Softwaremessung kann dazu beitragen, das Ziel zu erreichen

Vorbereitung und Durchführung

- Definition von Maßen
  - **Ziel:** Was soll durch die Messung erreicht werden?
  - **Fragen:** Was muss beantwortet werden, um das Ziel zu erreichen? Welche Eigenschaften müssen gemessen werden?
  - **Metriken:** Welche Zahlenwerte können erfasst werden, die die Eigenschaften widerspiegeln?
- Vorgehensweise beim Messen
  - **Messung:** Aufnahme der Zahlen aus dem Messobjekt
  - **Auswertung:** Aufbereitung der Zahlen in Bezug auf die untersuchten Fragen bzw. Eigenschaften
  - **Beurteilung:** Handlungsempfehlungen / Sollvorgaben



- „wahre Programmierer verachten Metriken“

Sichten / Messziele

- **Management**
  - Kosten der Software-Entwicklung (Angebot, Kostenminimierung)
  - Produktivitätssteigerung (Prozesse, Erfahrungskurve)
  - Risiken (Marktposition, Time2Market)
  - Zertifizierung (Marketing)
- **Entwickler**
  - Lesbarkeit (Wartung, Wiederverwendung)
  - Effizienz und Effektivität
  - Vertrauen (Restfehler, MTBF, Tests)
- **Kunde**
  - Abschätzungen (Budgettreue, Termintreue)
  - Qualität (Zuverlässigkeit, Korrektheit)
  - Return on Investment (Wartbarkeit, Erweiterbarkeit)

Prozedurale Komplexitätsmaße

- Umfangsmetriken
  - Dateigröße, LOC, NCSC usw.
  - Halstead-Metrik
  - Function Points
- Kontrollflussmetriken
  - Schachtelungstiefe
  - McCabe
- Datenstrukturmetriken
  - Variablenzahl, Records, Gültigkeitsdauern
- Stilmetriken
  - Namenskonventionen, Formatierung, Kommentierung
- Bindungsmetriken
  - Anzahl und Struktur der Bindungsbeziehungen

Halstead-Metriken

- Basisgrößen
  - η_1: Anzahl unterschiedlicher Operatoren (+,*, if,...)
  - η_2: Anzahl unterschiedlicher Operanden (Var, Const)
  - N_1: Gesamtzahl verwendeter Operatoren
  - N_2: Gesamtzahl verwendeter Operanden
- abgeleitete Größen
  - $\eta=\eta_1+\eta_2$: Größe des Vokabulars
  - $N=N_1+N_2$: Größe des Programms
  - $V=N*\log \eta$: Volumen
- Eigenschaften und Interpretation
  - $D=(\eta_1*N_2)/2\ \eta_2$: Schwierigkeit zum Schreiben oder Verstehen eines Programms

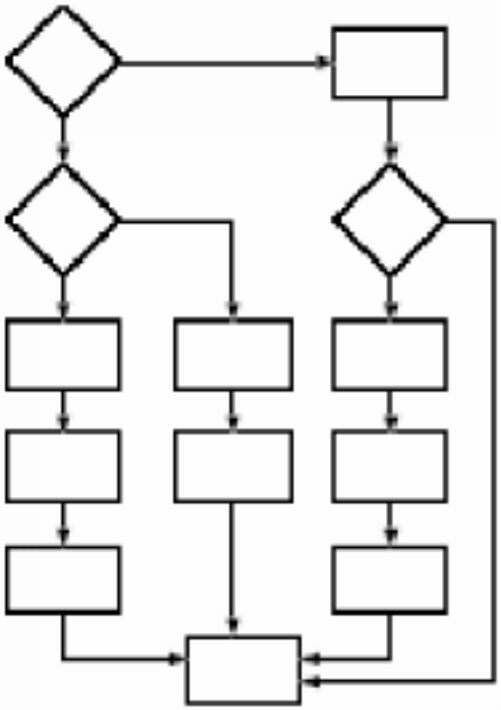
Schachtelungstiefe

- je höher die Schachtelungstiefe, desto schwerer die Verständlichkeit
- Maß: durchschnittliche Schachtelungstiefe
- z.B. auch: boolesche Anweisungen in Bedingungen!
- Auslagerung (Funktionen) erzwingt Struktur
- mit zunehmender Anzahl von Prozeduren und Tiefe des Schachtelungsbaumes wächst die Komplexität

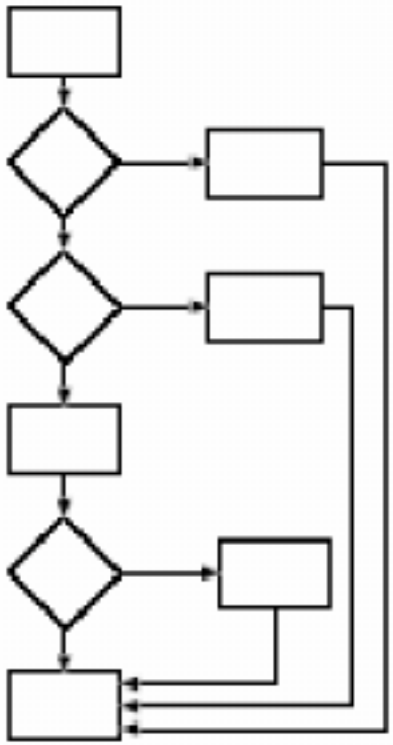
McCabe Metrik

- misst die strukturelle Komplexität von Programmen
- zyklomatische Zahl als Messgröße, aus dem Kontrollflussgraphen ermittelt
- Basisgrößen
  - e: Anzahl der Kanten des Kontrollflussgraphen
  - n: Anzahl der Knoten
  - p: Anzahl der verbundenen Komponenten
- zyklomatische Zahl
  - V(G)=e-n+2p
  - „Anzahl der Entscheidungen plus eins“

Beispiele



e = 15
n = 13
v(F) = 15 - 13 + 2



e = 11
n = 9
v(F) = 11 -9 + 2

Zyklomatische Zahl

- Die **zyklomatische Zahl** ist für strukturierte Programme um den Wert 1 größer als die Anzahl der binären Entscheidungen.
- Sie kann durch Abzählen der binären Entscheidungen ermittelt werden; existieren w Prädikate so gilt
  v(G) = w + 1 (nicht-binäre Entscheidungen mit n Ausgängen = n – 1 Binärentscheidungen)
- Empirische Untersuchungen: Fehlerhäufigkeit steigt überproportional falls zyklomatische Zahl > 10.
  - Empfehlung: maximale zyklomatische Zahl eines Moduls = 10
  - Überschreitet die zyklomatische Zahl eines Moduls diesen Wert, sollte Modul in Teilmodule aufgespalten werden


Software Metrics Glossary - McCabe & Associates - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Adresse http://www.mccabe.com/iq_research_metrics.htm
Wechseln zu
HOME › QUALITY MANAGEMENT › CONFIGURATION MANAGEMENT › ABOUT US › NEWS & EVENTS › SUPPORT › CONTACT US
McCabe Associates
Quality Management - McCabe IQ
"The development team used the tool to redesign functions that grew overly complex and reduced the effort and resources involved."
Manager, AIBI Advanced Software Construction Center
Software Metrics Glossary
McCabe IQ solutions use a vast number of software metrics in order to help you get the most precise assessment of your system's design and quality. We've defined these metrics below for your reference. Further details on many of these metrics can be found in the NIST document- "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric" by Arthur Watson and Tom McCabe.
McCabe Metrics
Cyclomatic Complexity Metric (v(G))
Cyclomatic Complexity (v(G)) is a measure of the complexity of a module's decision structure. It is the number of linearly independent paths and therefore, the minimum number of paths that should be tested.
Actual Complexity Metric (ac)
Actual Complexity (ac) is the number of independent paths traversed during testing.
Module Design Complexity Metric (iv(G))
Module Design Complexity (iv(G)) is the complexity of the design-reduced module and reflects the complexity of the module's calling patterns to its immediate subordinate modules. This metric differentiates between modules which will seriously complicate the design of any program they are part of and modules which simply contain complex computational logic. It is the basis upon which program design and integration complexities (S0 and S1) are calculated.
Essential Complexity Metric (ev(G))
Essential Complexity (ev(G)) is a measure of the degree to which a module contains unstructured constructs. This metric measures the degree of structuredness and the quality of the code. It is used to predict the maintenance effort and to help in the modularization process.
Pathological Complexity Metric (pv(G))
pv(G) is a measure of the degree to which a module contains extremely unstructured constructs.
Design Complexity Metric (S0)
S0 measures the amount of interaction between modules in a system.
Integration Complexity Metric (S1)
S1 measures the amount of integration testing necessary to guard against errors.
Object Integration Complexity Metric (OS1)
OS1 quantifies the number of tests necessary to fully integrate an object or class into an OO system.
Internet

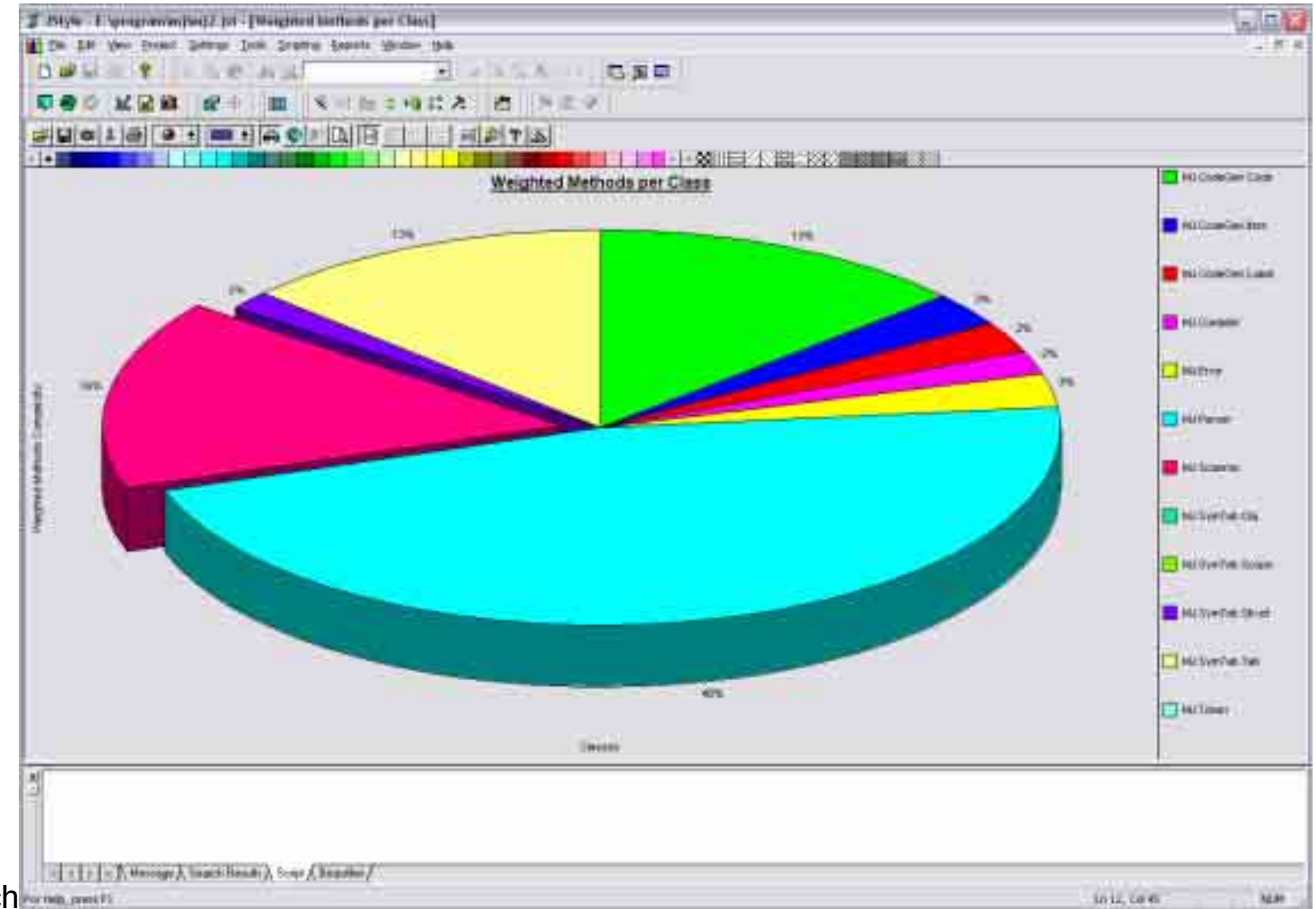
Objektorientierte Komplexitätsmaße

- CBO (coupling between objects)
  - Anzahl der Klassen, mit denen eine Klasse gekoppelt ist
- DIT (depth of inheritance tree)
  - Maximaler Weg von der Wurzel bis zur betrachteten Klasse
- NOC (number of children)
  - Anzahl der direkten Unterklassen; inverses Maß!
- RFC (response for a class)
  - Anzahl der Methoden, die potentiell ausgeführt werden können, wenn Objekt auf eingehende Nachricht reagiert
- ...

Werkzeugunterstützung

Werkzeugbeispiel JStyle 4.6

Quelle: Holzmann
www.ssw.uni-linz.ac.at/Teaching/ Lectures/Sem/2002/slides/Holzmann.**ppt**



Codierstandards (coding rules)

- prägen entscheidend die „Unternehmenskultur“ eines SW-Hauses
- „Freiheitsberaubung“ oder „corporate identity“?
- Namenskonventionen, Formatierung, Dokumentationsrichtlinien, ...
- zwingende Einhaltung gewisser Metriken (Modulgröße, Schachtelungstiefe, ...)
- automatisch überprüfbar
- meist ziemlich leicht zu „umgehen“
- verbessern im Allgemeinen die Produktqualität

Beispiel: Abraxas CodeCheck

Automating Corporate Source Code Compliance

Sample CodeCheck Coding Rules for C++

2.8.1 Use typedef names rather than the basic C types (int, long, float, etc.) for data members.

Example: In a program dealing with money, use

```
typedef float  Money;
Money          salary, bonus;  //   Good
```

rather than

```
float          salary, bonus;  //   Bad
```

Justification: This rule follows the principle of data abstraction. Typedef names do not increase the type safety of the code, but they do improve its readability. Furthermore, if we decide at a later time that money should be an int or a double we only have to change the type definition, not search the code for all float declarations and then decide which ones represent money.

Reference: Paragraph 4.21, XYZ C++ Guidelines

Beispiel: Java Conventions Rules

- http://java.sun.com/docs/codeconv/
- Why have code conventions? Code conventions are important to programmers for a number of reasons
  - 80% of the lifetime cost of a piece of software goes to maintenance.
  - Hardly any software is maintained for its whole life by the original author.
  - Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.

- Kapitel
  - File names, file organization, naming conventions
  - Indentation, comments, white space
  - Declarations, Statements, programming practices