otto group

Messen & Metriken
Ist Qualität messbar?

(André Fleischer)

prio.conference, Baden-Baden, 14.November 2007

Abstrakt

Seitdem Probleme mit Hilfe von Softwaresystemen gelöst werden, ist man auf der Suche nach Verbesserungen. Die Verbesserungen betreffen das Projekt- management, den Entwicklungsprozess und die eingesetzten Techniken. Um Verbesserungen feststellen zu können, müssen die Auswirkung von Veränderungen bestimmt werden. Diese Veränderungen können mit Hilfe verschiedener Kennzahlen, den Metriken, bestimmt bzw. gemessen werden. Für die Beurteilung von Prozessen, z.B. nach CMMI (Capability Maturity Model), ist das Steuern durch Metriken unerlässlich. Doch wie sieht dies auf der technologischen Seite aus und wie kann man Verbesserungen und die Qualität der Software messen? Dieser Vortrag stellt eine Auswahl von Metriken vor, zeigt deren Praxisrelevanz und skizziert deren Einsatz, sowie Vorgehensweisen, wie mit Hilfe von Metriken Software-Qualität gesteigert werden kann.
Der Vortrag zeigt anhand praktischer Erfahrungen wie Metriken gebildet werden können, damit der Teilnehmer "gleich morgen" mit dem Thema Metriken starten kann, um die Qualität seiner Software zu steigern.

André Fleischer, otto group

1995 – 2000	**Studium Technische Informatik (HAW Hamburg)**
2001 – 2007	**Lufthansa System** **IT Consultant, Software Architekt**
Seit 2007	**Otto Group, Software Architekt**
Focus	**Java, JEE, Software-Architekturen,** **Software-Entwicklungs-Prozesse,** **Objektorientierte Vorgehensweisen,**
Kontakt	andre.fleischer@ottogroup.com http://www.xing.com/profile/andre_fleischer

Überblick

- **Motivation**
- **Metriken**
- **Werkzeuge**
- **Beispiele aus der Praxis**
- **Zusammenfassung**

Motivation – Unsere Probleme

- Die Kontrolle der technischen Qualität und Struktur von Software ist schwierig, ihre Durchsetzung sogar noch schwieriger

- Erosion von Struktur und Architektur ist ein allzu bekanntes Phänomen
  - Systemwissen ist praktisch immer ungleich verteilt, Know-How-Defizite
  - Die Systemgröße erzeugt eine zusätzliche Dimension der Komplexität
  - Unbemerkt werden unerwünschte zyklische Abhängigkeiten geschaffen
  - Kopplungsgrad und Komplexität wachsen schnell (und vorerst unbemerkt) über ein noch beherrschbares Niveau
  - Zeit- und Kostendruck ist stets ein guter Grund um Struktur zu opfern
  - Anwendungs-Domäne vs. Technische Domäne

- Typische Symptome einer erodierten Architektur sind ein hoher Kopplungsgrad sowie zyklische Abhängigkeiten
  - Änderungen werden schwieriger
  - Es wird schwieriger, das System zu testen oder zu verstehen
  - Deployment-Probleme aller Art …

Qualität?

Softwarequalität nach DIN 55350 :
Die Gesamtheit der Eigenschaften oder Merkmale, die Software in Verwendung und (Weiter-) Entwicklung aufweist, um die an sie gestellten Anforderungen zu erfüllen.

Innere & Äußere Qualität Kriterien

- Innere Software-Qualität:
  - Qualität aus Sicht des Software-Erstellers
  - Wartbarkeit
  - Erweiterbarkeit
  - Wiederverwendbarkeit
  - Verständlichkeit

- Äußere Software-Qualität
  - Qualität aus Sicht des Anwenders
  - Fachliche Korrektheit

➔ Gute Innere Struktur und Qualität führt zu guter äußere Qualität

Audits, Metriken & Berichte

Wie kann ich die Einhaltung meiner Architektur prüfen?

Wie kann ich meine Quelltexte prüfen?

Was sind meine Qualitätskriterien?

- **Audits: Prüfung der Software durch Auditoren**
- **Metriken: Vermessung der Software**
- **Berichte: Publizierung von Ergebnissen**

Audits, Metriken & Berichte

- Audits: Prüfung der Anforderungen an die Software und deren Bestandteile
- Funktionale Prüfung:
  - Äußere Qualität durch Anwender- und Integrationstests
- Technische Prüfung:
  - Innere Qualität durch Metriken und Regeln

- Fehlende Akzeptanz der Audits
  - Fehlende Regeln und Standards
  - Keine klaren Kriterien
  - Drei Experten mit Drei Meinungen im Audit

Gutes Design!?

- Abstraktion
  - Schaffung einfacher Sichten auf Konzepte
- Modularisierung
  - Zerlegung der Komplexität in handhabbare Einheiten
  - Kapselung
  - Lose Kopplung zwischen Subsystemen
  - Hohe Kohäsion in den Subsystemen
- DRY – Don‘t Repeat Yourself
  - Wiederverwendung zur Reduzierung und Vereinfachung
- KISS: Keep It Stupid Simple (A. Tanenbaum)
  - So komplex wie nötig, so einfach wie möglich
- Millers Law: 7 ± 2

Überblick

- **Motivation**
- **Metriken**
- **Werkzeuge**
- **Beispiele aus der Praxis**
- **Zusammenfassung**

Metriken im Sport

Richtig Schätzen mit dem Richtigen Werkzeug

Bestimmen Sie das Gewicht!

Nur anhand eines Fotos?

Sie haben doch Ihre Augen!
Die Angabe Gramm genau!

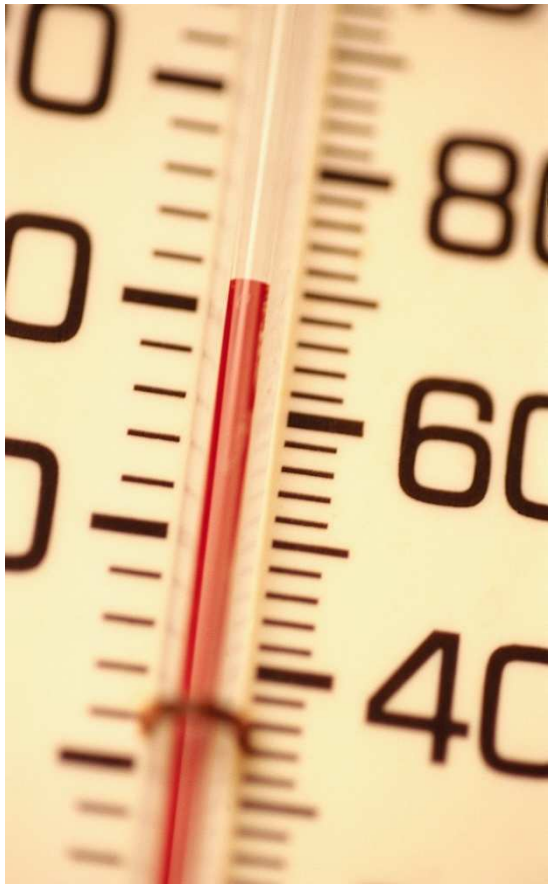
Richtig Schätzen mit dem Richtigen Werkzeug

Wie genau muß gemessen werden?

Passende Kenngröße?

Passende Instrumente?

Messen in der Geschichte



- 2000 v.Chr. „…heißer als…“
- 1600 Erste Thermometer, Galileo Galilei
- 1724 Fahrenheit Skala
- 1742 Celsius Skala
- 1954 Absoluter Nullpunkt, Kelvin Skala

- Bestimmung der Größe: Messinstrument
- Numerische Abbildung: Skala
- Vergleichsmöglichkeit: Größer als, Kleiner als
- Direkte Bestimmung
- Indirekte Bestimmung: Über Beziehungen

Software Measurement ?!?

„You cannot control what you cannot measure“
DeMarco 1982

Metriken – Vermessen der Eigenschaften des Software-Systems

- „Die **Metrik** (griechisch μετρική - Zählung, Messung) bezeichnet im Allgemeinen ein System von Kennzahlen oder ein Verfahren zur Messung einer quantifizierbaren Größe.“ (Wikipedia.de, 2007)

- Speziell für Softwareentwicklung: Algorithmus zur Berechnung von Kennzahlen eines Software-Systems

- Ziel der Software Metrik:
  - Beurteilung der Elemente
  - Schwachstellensuche
  - Fehlerprognose
  - Basis der Aufwandschätzung
  - Projektentwicklung verfolgen
  - Entscheidungen steuern

Metriken – Vermessen der Eigenschaften des Software-Systems

- Wie kann man nachweisen, daß Änderungen den gewünschten Erfolg haben?
  - Prozess-Änderungen
  - Neue Funktionalität
  - Neue Methodiken
  - Neue Technologien
  - Iterationen

- Bewertung und Beurteilung von:
  - Entwicklungsphasen
  - Phasenergebnissen
  - Technologien
  - Abnahmen von Zulieferungen
- Risikoabschätzung

Metriken – Vermessen der Eigenschaften des Software-Systems

- Management: Kosten, Produktivität, Risiken
- Entwickler: Lesbarkeit, Effizienz, Vertrauen
- Kunde: Abschätzungen, Qualität, ROI

- Prozessmetriken: Ressourcen, Fehler, Kommunikationsaufwand
- Produktmetriken: Umfangmetrik (Softwaregröße, Fertigstellungsgrad, LOC), Komplexität, Lesbarkeit, Entwurfsqualität, Produktqualität, Antwortzeiten, Anforderungsqualität
- Projektmetriken: Auftragsvolumen, Teamgröße, Dauer
- Aufwands/Kosten-Metriken: Aufwandsstabilität, Aufwandsverteilung, Produktivität, Aufwand/Termin
- Zeitmetrik: Entwicklungszeit, Meilenstein-Trends, Termintreue
- Geschäftsmetrik: Schulungsgrad, Kundenzufriedenheit

Metriken – Vermessen der Eigenschaften des Software-Systems

- Ich brauche eine Möglichkeit, die logische Architektur eines Systems auf einfache Art und Weise abstrakt zu beschreiben
- Nur eine möglichst kleine Auswahl relevanter Metriken zur Überwachung der technischen Qualität
- Klare Definition einer Metrik notwendig:
  - Instrument, Skala, Vergleich, Schwellwerte
- Wie interpretiere ich die Ergebnisse? Welcher Wert ist gut?

Definition von Regeln um die Probleme & Fragestellungen näherungsweise zu lösen (Heuristik)

Metriken – Ein Beispiel

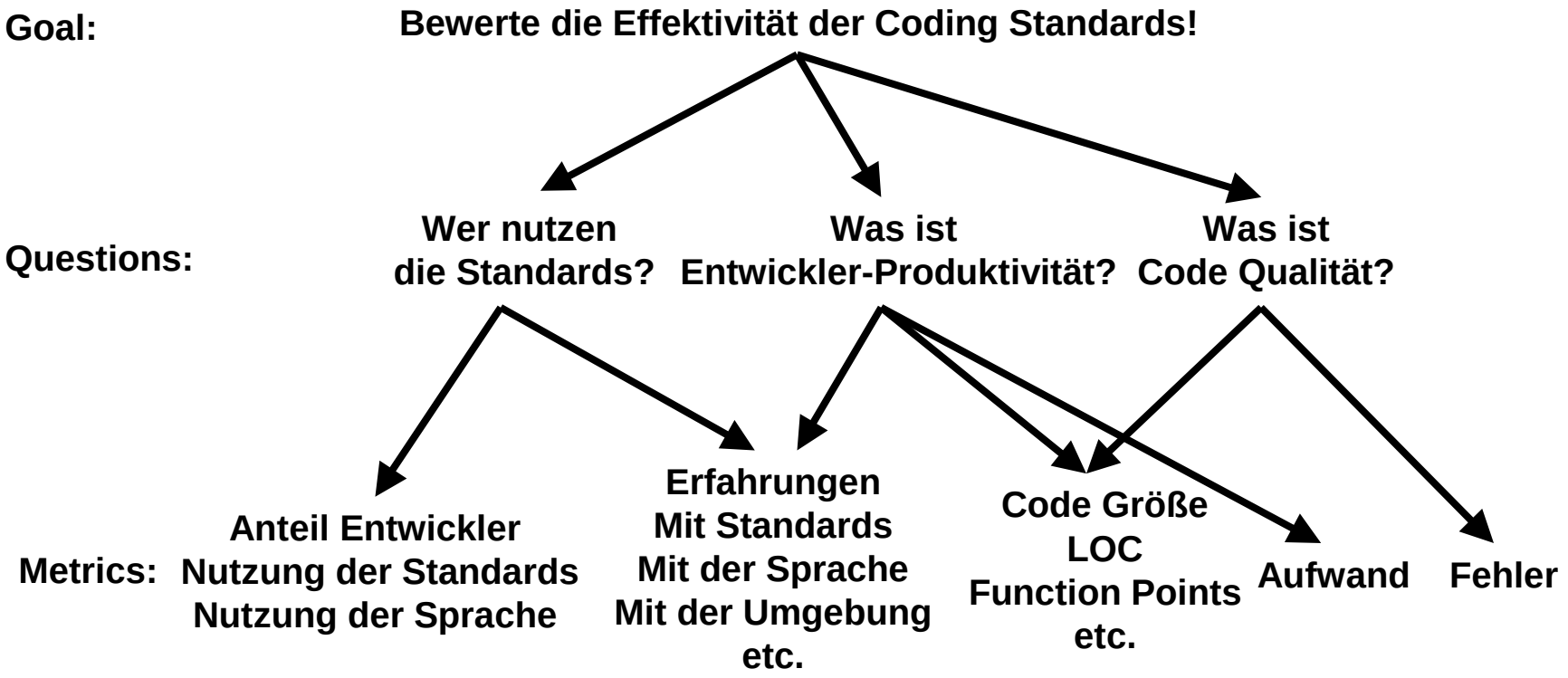
- Forderung:
  - Modularisierung, lose Kopplung bei Vererbung
  - Schlanke Schnittstellen
- Metriken:
  - Messung der Vererbungstiefe
  - Messung Anzahl Parameter
- Definition von Schwellwerten:
  - Vererbungstiefe < 8
  - Parameterzahl < 6

- Was habe ich erreicht?
  - Ich nehme an, daß mein Design gut ist, wenn ich weniger als 8 Vererbungstiefen habe.
  - Ich nehme an, daß meine Schnittstellen gut sind, wenn ich weniger als 6 Parameter übergebe

Qualitätsmodelle

- **Nach Boehm, 1978**
  - Portabilität
  - Zuverlässigkeit
  - Effizienz
  - Testbarkeit
  - Verständlichkeit
  - Veränderbarkeit
- **Nach McCall, 1977**
  - U-F-C-M, Use-Factor-Criteria-Metric
  - Usability
  - Integrierbarkeit
  - Effizien
  - Korrektheit
  - Zuverlässigkeit
  - Wartbarkeit
  - Testbarkeit
  - Flexibilität
  - Wiederverwendbarkeit
  - Portabilität
  - Zusammenarbeit

Goal-Question-Metric



Goal-Question-Metric

- Verbindung zwischen den (Qualitäts-) Zielen des Projektes mit den Metriken

- Welche Teile der Anwendung werden von welchen anderen Teilen verwendet?
- Werden die Architekturegeln eingehalten?
  - ➔ Kopplung, Zyklen
- Welche Teile werden nicht – mehr – verwendet?
- Welche Teile werden intensiv genutzt ?
  - ➔ Änderungen haben große Auswirkungen
- Welche Fremd-Bibliotheken werden verwendet?
  - ➔ Gewollte Verwendung, Redundanzen, Lizenzen
- Wie hoch ist meine Testabdeckung?
- Wie hoch ist der Grad der Quellcode-Dokumentations
- Werden die Programmierrichtlinien eingehalten?
- Sind Code-Duplizierung vorhanden?

Metriken – Vermessen der Eigenschaften des Software-Systems

- Management: Kosten, Produktivität, Risiken
- **Entwickler: Lesbarkeit, Effizienz, Vertrauen**
- Kunde: Abschätzungen, Qualität, ROI
- Prozessmetriken: Ressourcen, Fehler, Kommunikationsaufwand
- **Produktmetriken: Umfang (LOC), Komplexität, Lesbarkeit, Entwurfsqualität, Produktqualität, Anforderungsqualität**
- Aufwands/Kosten-Metriken: Aufwandsstabilität, Aufwandsverteilung, Produktivität, Aufwand/Termin
- Zeitmetrik: Entwicklungszeit, Meilenstein-Trends, Termintreue
- **Umfangmetrik: Softwaregröße**, Fertigstellungsgrad
- Geschäftsmetrik: Schulungsgrad, Kundenzufriedenheit

Verschiedene Metriken

- NCSS:
  - Non Commenting Source Statements,
  - Zählen von ; und {
- Kommentare:
  - Zählen der Zeilen // und /*..*/
- Anzahl
  - Klassen,
  - Abstrakter Klassen,
  - Interfaces,
  - Methoden
- Durchschnittswerte
- ACD: Average Component Dependency,
  - Messung der Kopplung,
  - Anzahl Abhängigkeiten

Verschiedene Metriken

- McCabe & Halstead: Cyclomatic Complexity Number (CCN)
  - Anzahl der möglichen Ablaufpfade durch eine Methode
  - Kennzahl zur Abschätzung der Komplexität
  - Anzahl der Ablaufpfade „if, for, switch“ ,
  - Anzahl notwendiger Testfälle
  - Aber: Nicht alles mit einer hoher CCN ist unverständlich, z.B. switch…

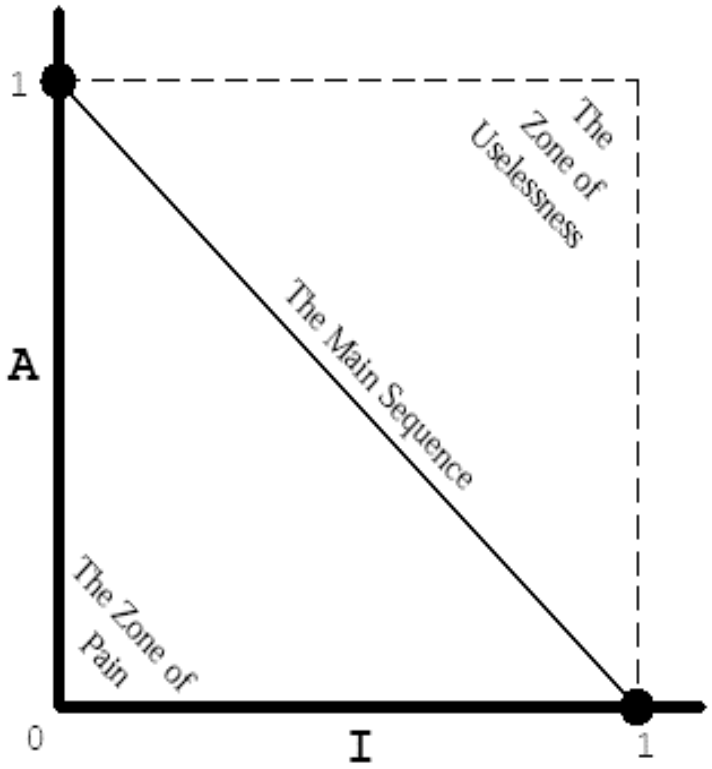
1-10	A simple program, without much risk
11-20	more complex, moderate risk
21-50	complex, high risk program
> 50	untestable program (very high risk)
http://www.sei.cmu.edu/str/descriptions/cyclomatic body.html	

  - Einsatz z.B. bei Qualitätssicherung für Software des Zugtunnels:

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

Verschiedene Metriken

- Ca: Afferent Coupling
  - Eingehende Abhängigkeiten
- Ce: Efferent Coupling
  - Ausgehende Abhängigkeiten
- Instabilität:
  - I = Ce / (Ce + Ca)
  - 0<I<1, 0=stabil
- Abstraktheit:
  - A= Abstract Classes/All Classes
  - 0<A<1, 0=Konkret, 1=Abstrakt
- Distanz:
  - D= A+I-1
  - Je Abstrakter umso stabiler



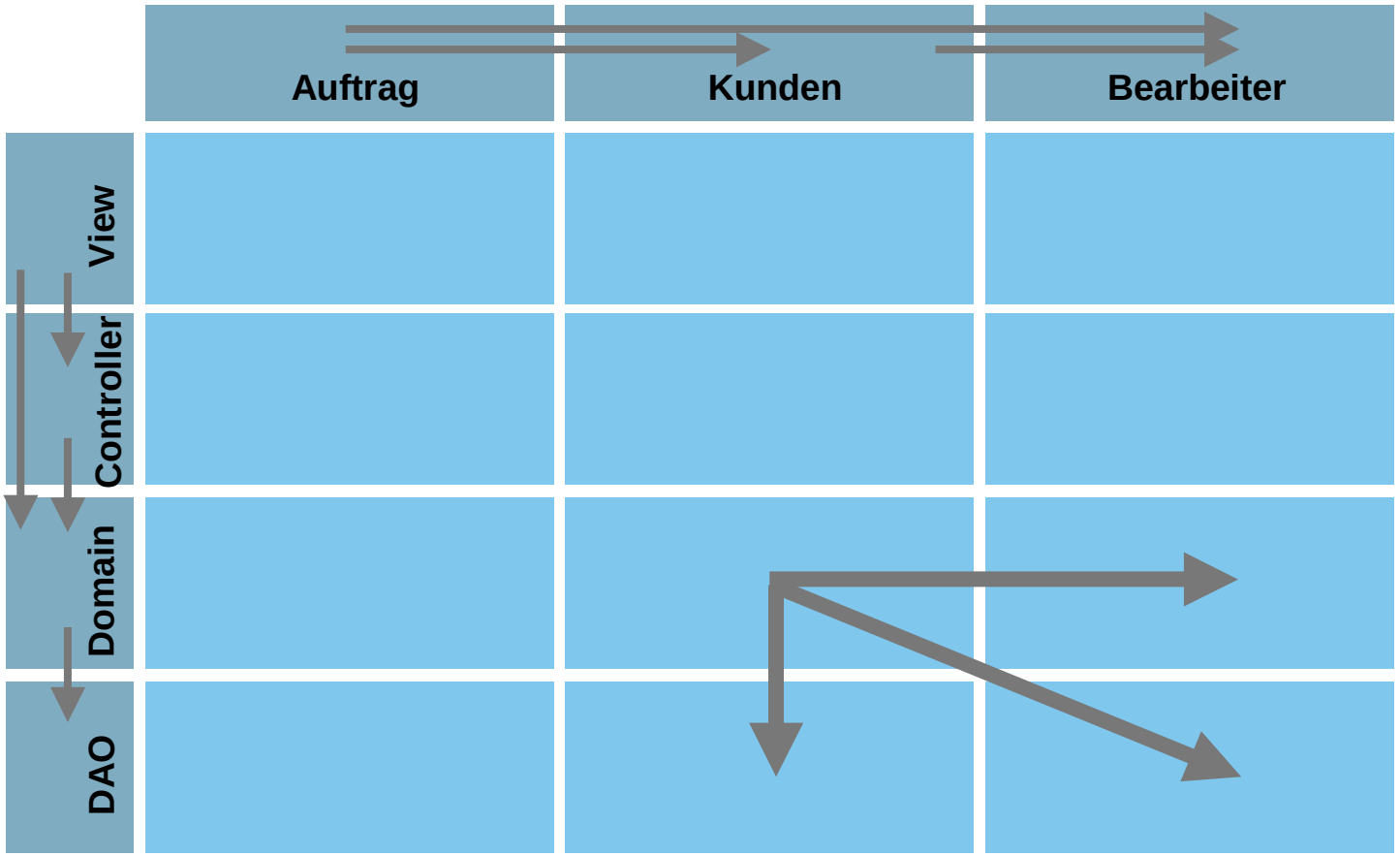
Chidamber & Kemerer

- Weitere OO-Metriken nach Chidamber und Kemerer
  - WMC - weighted methods per class
  - DIT - depth of inheritance tree
  - NOC - number of children
  - CBO - coupling between objects (Ca, Ce)
  - RFC - response for a class (Anzahl eigener plus Anzahl aufgerufener Methoden)
  - LCOM - lack of cohesion in methods

Logische Architektur
Erlaubte Zugriffspfade

Logische Architektur

Abhängigkeitspfade



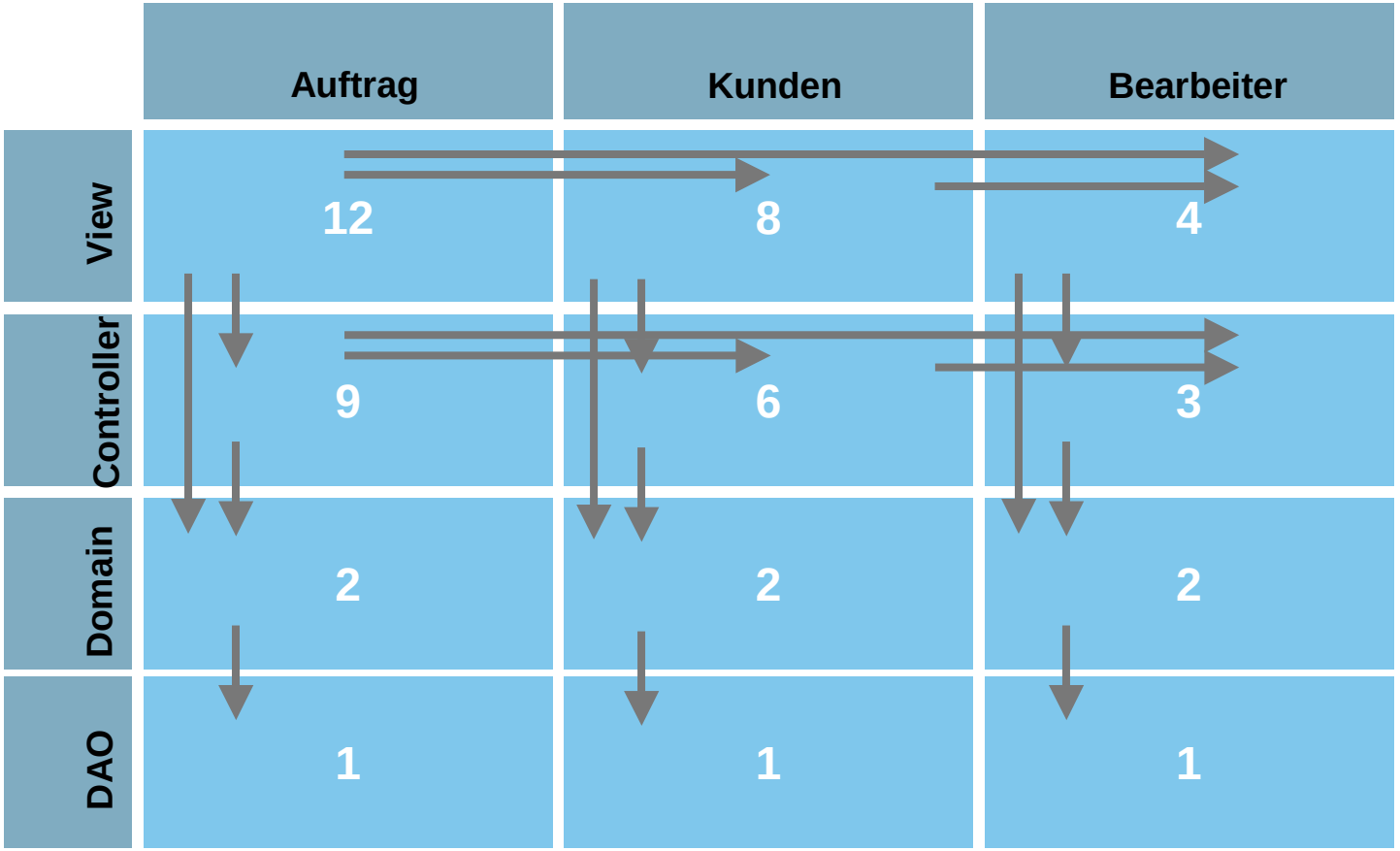
Average Component Dependency

Average Component Dependency

ACD=60/12=5

Average Component Dependency

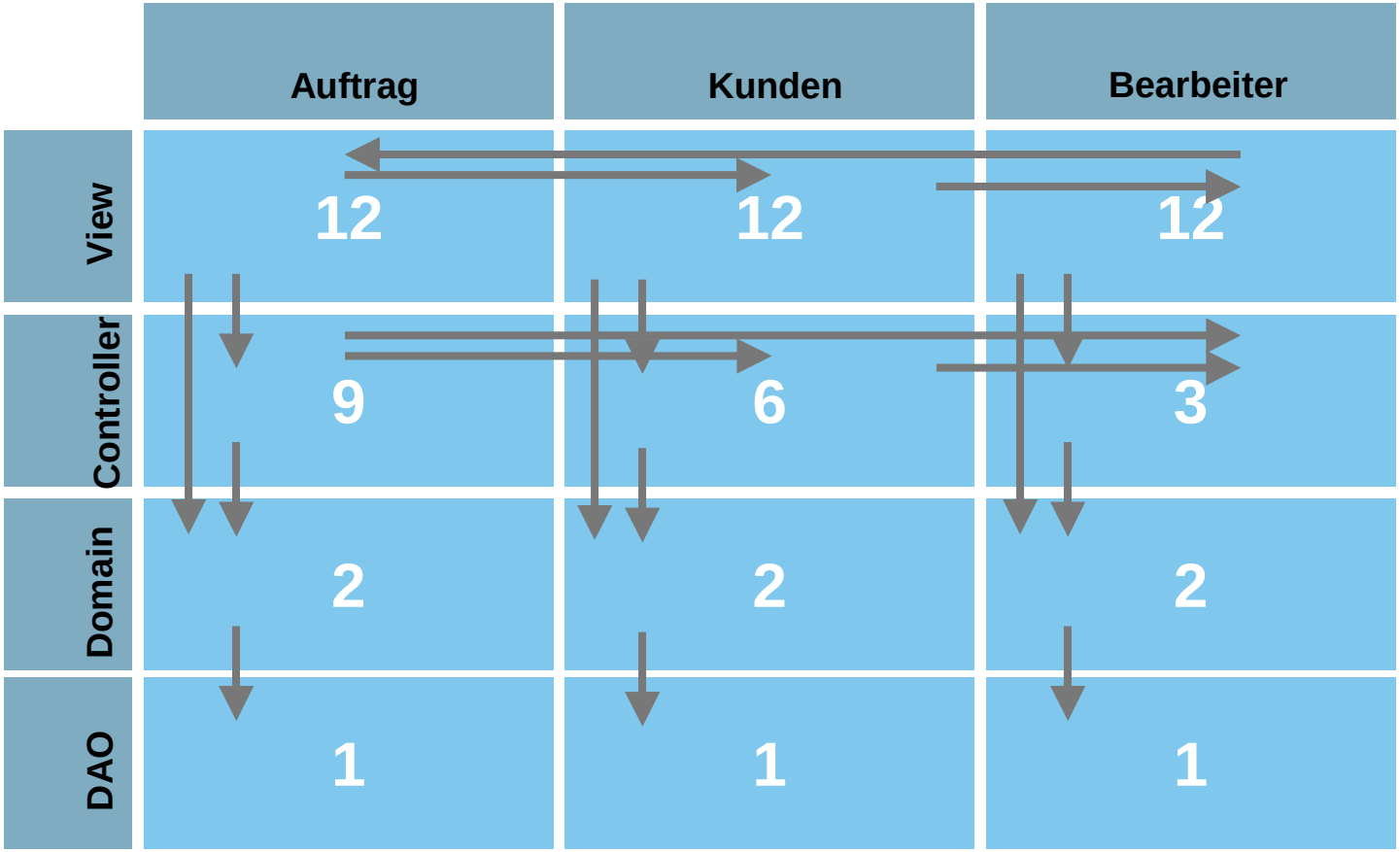
ACD=51/12=4,25 **rACD=15%**



Average Component Dependency

ACD=63/12=5,25

rACD=~20%



otto group

Logische Architektur

- Jedes Subsystem gehört zu genau einer Schicht
- Ein Subsystem kann zu einer fachlichen Komponente gehören
- Zuordnung zwischen fachlicher Komponente und Subsystemen erfolgt am besten über Namenskonventionen
- Fachliche Komponenten müssen nicht unbedingt auf jeder Ebene vertreten sein
- Subsysteme können „privat“ sein
- Technische Subsysteme können oft keiner fachlichen Komponente zugeordnet werden
- Sehr technische Systeme haben oft nur Schichten, aber keine fachlichen Komponenten

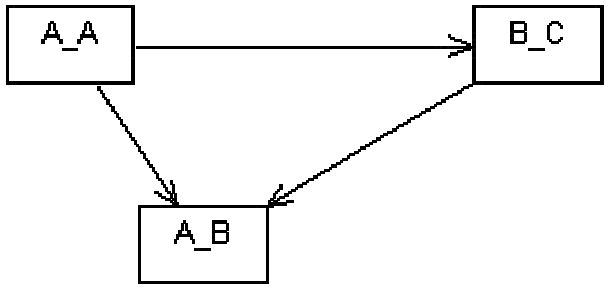
- Die logische Architektur muss anhand definierter Regeln in eine physische Implementierung überführt werden

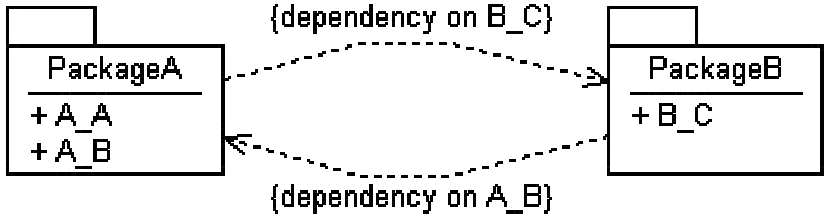
Zyklische Abhängigkeiten

- Zyklen wirken sich negativ auf die Qualität aus
- Erhöhter Kopplungsgrad ➔ Hoher ACD
- Zyklenteilnehmer können nicht mehr getrennt getestet werden
- Verständlichkeit für den Quelltext wird reduziert, Teilnehmer müssen insgesamt verstanden werden
- Zyklen erhöhen die Komplexität
- Auflösung durch Dependency Inversion und Interfacing
- Prüfung mit Distance-Metrik

Zyklische Abhängigkeiten

- Es sollte keine Zyklen innerhalb der Abhängigkeits-Struktur gegeben
- Paket A und B müssen immer gleichzeitig entwickelt, implementiert bzw. gewartet werden
- Zyklen müssen durchbrochen werden ...

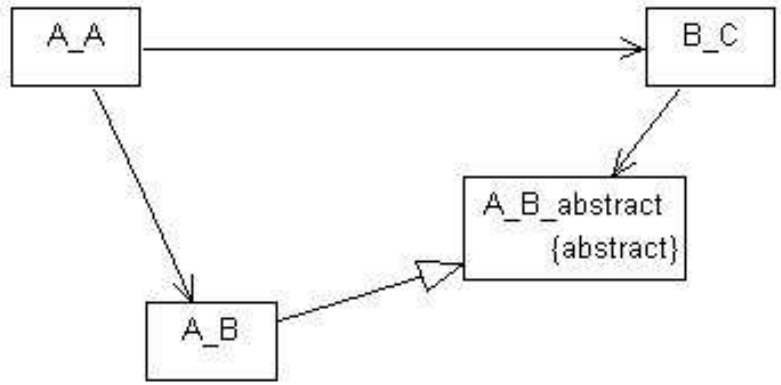


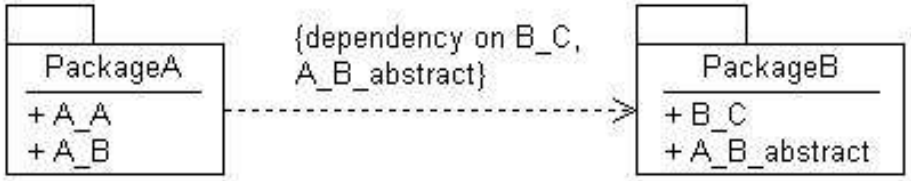


Beide Packages sind gegenseitig voneinander abhängig.

Zyklische Abhängigkeiten

- Durch die Einführung einer neuen Abstraktion wurde die zyklische Abhängigkeit unterbrochen.





Es besteht nun nur noch eine einseitige 'basiert auf'- Beziehung.

Gefahren durch Metriken?

- Komplexe Software wird auf Zahlen reduziert
- Sinnlose und Sinnvolle Metriken
- Illusion of Simplicity, Verlagerung von Komplexität
- Interpretation der gemessenen Zahlen
- Viel hilft viel
- Jedes Projekt ist anders
- Psychologische Konsequenzen

➔ Auswahl von Metriken nach GQM
➔ Dokumentation der Metriken und Schwellwerte
➔ Realistische Schwellwerte
➔ Begründete Ausnahmen
➔ Metriken liefern nur Hinweise

Überblick

- **Motivation**
- **Metriken**
- **Werkzeuge**
- **Beispiele aus der Praxis**
- **Zusammenfassung**

Werkzeuge

- Werkzeuggestützte Analysen können…
  - …die automatischen Metriken liefern
  - …helfen Risiken identifizieren
  - …Fakten für Entscheidungen und Diskussionen liefern
  - …nach faule Kompromisse im Design suchen
  - …sind die einzige Möglichkeit objektive Antworten zugeben
  - …regelmäßig und wiederholbar ausgeführt werden

- Was ist mit manuellen Metriken?

Welche Tools?

- PMD: Regelbasierter
  - Java Quellcode Analyzer
  - Mögliche Fehlerquellen: String Verwendung, Logging, Doppelter Code, Dead Code etc.
- Checkstyle
  - Quellcode auf Einhaltung von Code Conventions prüfen
  - Kommentierung, Naming Convention, Länge, Einrückung, CodeFlaw etc.
- JDepend/NDepend
  - Liefert Metriken zu Code-Abhängigkeiten
  - Zyklen, Concreteness, Abstracness, Kopplung
- JavaNCSS und Verwandte
  - Bestimmt die LOC im System
  - NCSS, CCN für Gesamt, Package, Klassen, Funktion
- Sun JavaAVK
  - Prüft Anwendung auf J2EE-Kompatibilität

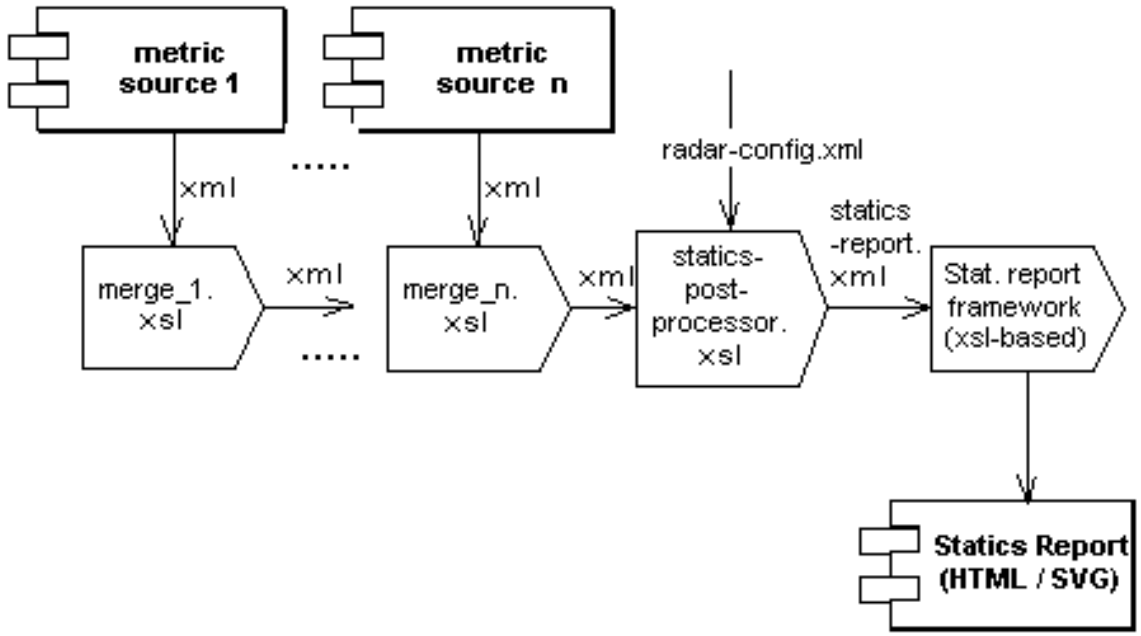
Welche Tools?

- Sotograph:
  - Architektur-Definitions- und Überwachungswerkzeug,
  - Kommerziell
- SonarJ
  - Architektur-Definitions- und Überwachungswerkzeug,
  - Kommerziell
- XRadar:
  - Architektur-Definitions- und Überwachungswerkzeug,
  - OpenSource,

- IBM Rational Software Architect: Analyse, Audits, Metriken
- IBM Rational ClearCase: Zulieferung von Metriken, Anzahl Veränderungen
- IBM Rational ClearQuest: Zulieferung von Metriken, Anzahl Issues, Dauer von Änderungen
- IBM Rational ProjectConsole: Aggregation von Informationen aus Rational Plattform

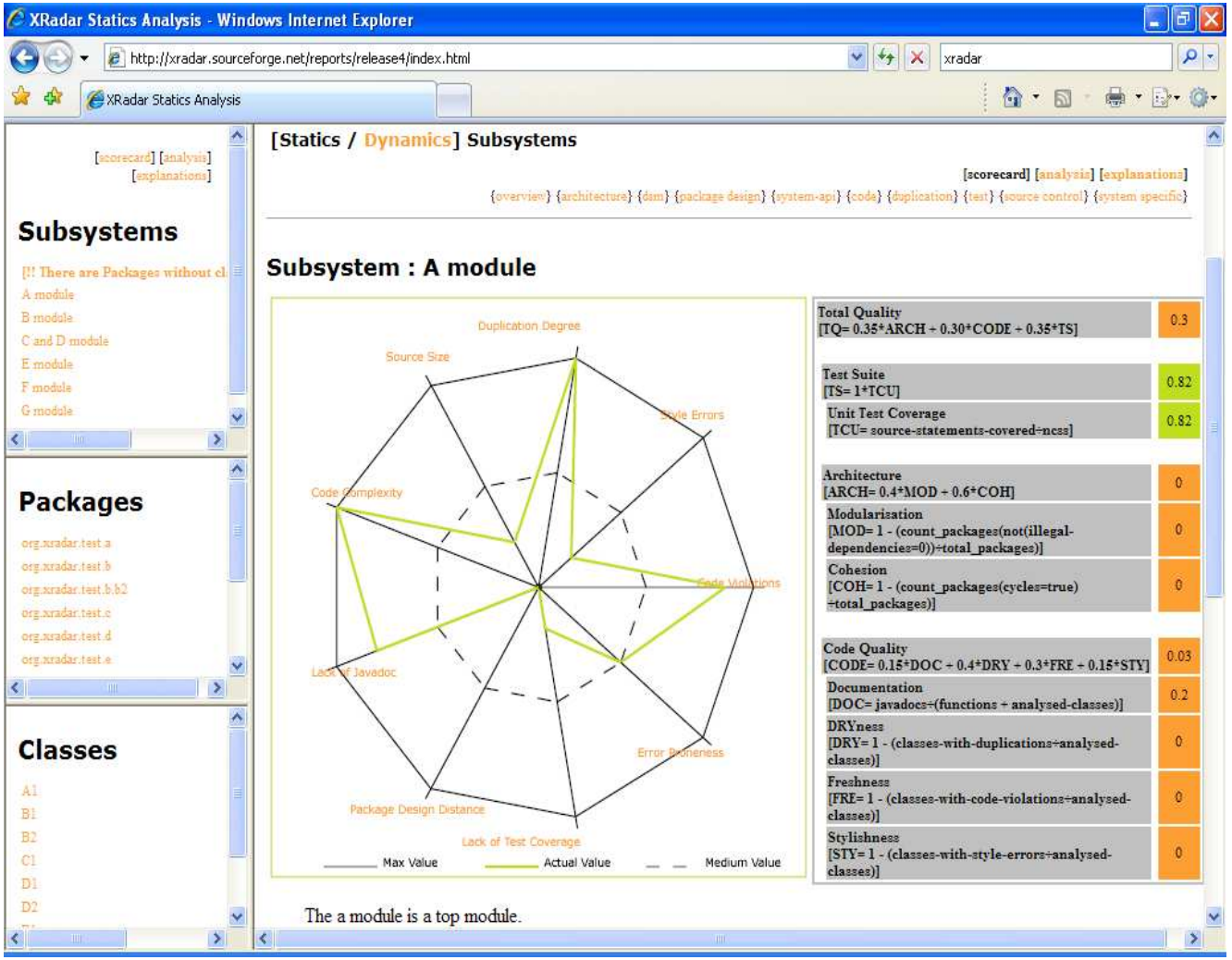
XRadar

- Besteht aus ant / maven targets
- Basiert auf mehreren Open-Source Werkzeugen
  - JavaNCSS, JDepend, JUnit, Cobertura, Checktyle, PMD
- Zusammenführung durch XSL-Transformationen
- Erweiterbar durch weitere Werkzeuge und eigene XML-Quellen, XSL-Transformationen



Quelle: XRadar

XRadar Demo



otto group

Überblick

- **Motivation**
- **Metriken**
- **Werkzeuge**
- **Beispiele aus der Praxis**
- **Zusammenfassung**

Projekte im Vergleich

Metrik	Konzern-Projekt	Logistik-Projekt	JDK6 java.*
# of Classes	4.853	6.951	1.510
LOC overall	1.293.011	~1.100.000	215.690
Comments	387.457	~65.000	17.011
Methods per Class	11,05	9,83	12,42
NCSS non generated	349.658	430.822	142.895
NCSS per Class	72,05	61,98	94,63
NCSS per Method	6,52	6,31	7,62
CCN	2,06	Unbekannt	2,91

Überblick

- **Motivation**
- **Metriken**
- **Werkzeuge**
- **Beispiele aus der Praxis**
- **Zusammenfassung**

Quellen

- PMD: http://pmd.sourceforge.net/
- Checkstyle: http://checkstyle.sourceforge.net/
- JavaNCSS: http://www.kclee.de/clemens/java/javancss/
- JDepend: http://clarkware.com/software/JDepend.html
- XRadar: http://xradar.sourceforge.net/
- SonarJ: http://www.hello2morrow.de/angebot/sonarj.php
- AVK: http://java.sun.com/j2ee/avk/
- SA4J: http://www.alphaworks.ibm.com/tech/sa4j

- Metriken im Projekt: http://www.pragmaticprogrammer.com/sk/auto/
- Literatur: Software Metrics A Rigorous and Practical Approach ISBN: 0534954251
  - Mathematische Hintergrund: http://www.dissertation.de/index.php3?active_document=/FDP/sj929.pdf
- JavaMagazin 02.2007: Architekturmanagement
- ObjektSpektrum 03.2007: Metriken im praktischen Einsatz

  http://www.sigs-datacom.de/sd/publications/pub_article_show.htm?&AID=2064&Table=sd_article

Quellen

- Metriken im Projekt: http://www.pragmaticprogrammer.com/sk/auto/
- Literatur: Software Metrics A Rigorous and Practical Approach ISBN: 0534954251
  - Mathematische Hintergrund: http://www.dissertation.de/index.php3?active_document=/FDP/sj929.pdf
- JavaMagazin 02.2007: Architekturmanagement
- ObjektSpektrum 03.2007: Metriken im praktischen Einsatz

  http://www.sigs-datacom.de/sd/publications/pub_article_show.htm?&AID=2064&Table=sd_article

Grenzen

- Automatisch erzeugte Analysen sind ehrlich:
  - Objektiv, wiederholbar, unbestechlich
  - Ergebnisse immer noch prüfen
  - Kein Absolutheitsanspruch
  - ➔Metriken, Regeln, Schwellwerte immer anpassen

- Analysen verändern die Arbeitsweise:
  - „Hilfe ich werde begutachtet“
  - Programmieren für die Regeln
  - Metriken & Analysen erklären, Nutzen aufzeigen
  - Langsam anfangen
  - ➔ Transparenz schaffen

  - Abstumpfung durch Ergebnisflut
  - Probleme bei der Interpretation
  - Korrektheit der Annahmen
  - Fehlende Konsequenzen

Zusammenfassung
Wie soll ich anfangen?

- Ziel: Audits, Metriken und Berichte
- Ehrlichkeit und Mut!

- Ziel der Analysen definieren
- Wer? Was? Wie Oft? Wann?
- Überzeugungsarbeit leisten und Sponsor finden!
- Build-Prozess automatisieren!
- Logische Architektur definieren und aufmalen!

- Definition von Regeln und Standards
- Audits durchführen (Geht auch ohne Werkzeuge)
- Maßnahmen laut Audits durchführen

- Welche Analysen kann ich mit vorhandenen Tools machen?
- Auswahl Metriken und Definition von Schwellwerten
- Falls notwendig ein Werkzeug einführen

Regeln für das Projekt

- Klar definierte, zyklenfreie, logische Architektur
- Definition der Abhängigkeiten zwischen Schichten und möglichst der fachlichen Komponenten
- Der Quelltext muss diese logischen Regeln abbilden
- Zyklen zwischen Sub-Systemen sind nicht erlaubt
- Der Kopplungsgrad ACD ist niedrig zu halten
- Codezeilen < 500 für Übersetzungseinheit (Klasse) auf Basis von NCSS
- Zyklomatische Komplexität < 25 pro Methode

Täglicher Projekt-Report

Q & A

andre.fleischer@ottogroup.com

http://www.xing.com/profile/andre_fleischer