



**syngenio**  
Aktiengesellschaft

We make IT work.

---

## **Bessere Code-Qualität mit zusätzlichen Tools/Eclipse-Plugins**

---

Referent Heiko W. Rupp  
Stuttgart, 07.07.2005

## Über mich

- n Diplom Informatiker
  - n Universität Karlsruhe
- n Wohnt und arbeitet in Stuttgart
- n 20 Monate alter Sohn
- n Aktiv in der Open Source
  - n JBoss
  - n Xdoclet
  - n Früher NetBSD
- n JBoss-Buch bei dpunkt.verlag erschienen



## Agenda

- n Motivation – um was geht es?
- n Kategorisierung und Beispiele
- n Werkzeug CAP
- n Werkzeug Checkstyle
- n Werkzeug Findbugs



## Motivation – um was geht es?

- n Code Qualität
  - n Weniger Fehler zur Laufzeit
  - n Leichtere Lesbarkeit
  - n Höhere Wiederverwendbarkeit
  - n Geringerer Speicherverbrauch



syngenio

We make IT work.

## Drei Kategorien

- n Analyse nach Metriken
  - n Auf Quellcode-Ebene
  - n z.B. JDepend, CAP
- n Stil-Analyse
  - n Auf Quellcode-Ebene
  - n z.B. Checkstyle, PMD
- n Programmfluss-Analyse
  - n Auf Ebene des Bytecodes
  - n z.B. Findbugs

## Analyse nach Metriken

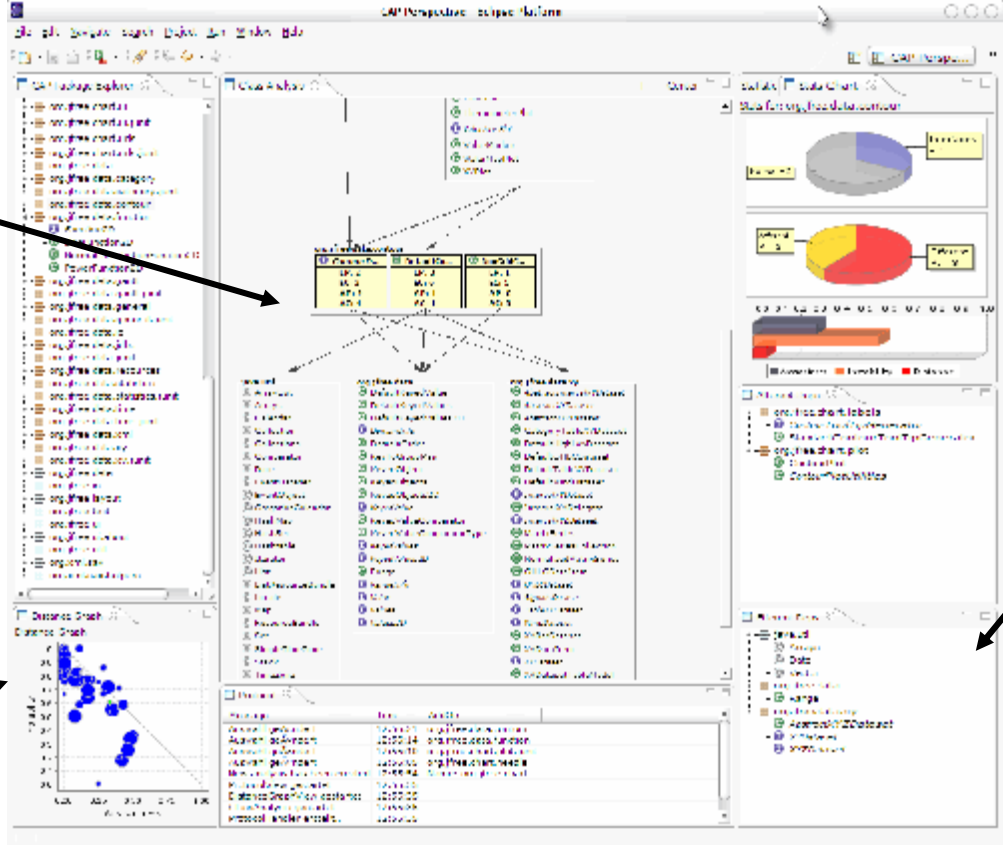
- n Höhere Wiederverwendbarkeit
- n Viele Abhängigkeiten
  - n -> schlechte Wiederverwendung
  - n -> höhere Wartungskosten
- n Metriken beschreiben die Abhängigkeiten
  - n Paper von Robert Martin (\*)
- n Ziel:
  - n Lose Kopplung zwischen Paketen

(\*) <http://www.objectmentor.com/resources/articles/oodmetric.pdf>

## CAP – Code Analysis Plugin

- n <http://cap4e.sourceforge.net/> alias <http://cap.xore.de/>
- n UpdateSite <http://cap.xore.de/update>
- n Benötigt Draw2D aus GEF-Paket
  
- n Eigene Perspektive für grafische Auswertungen

# CAP – Perspektive



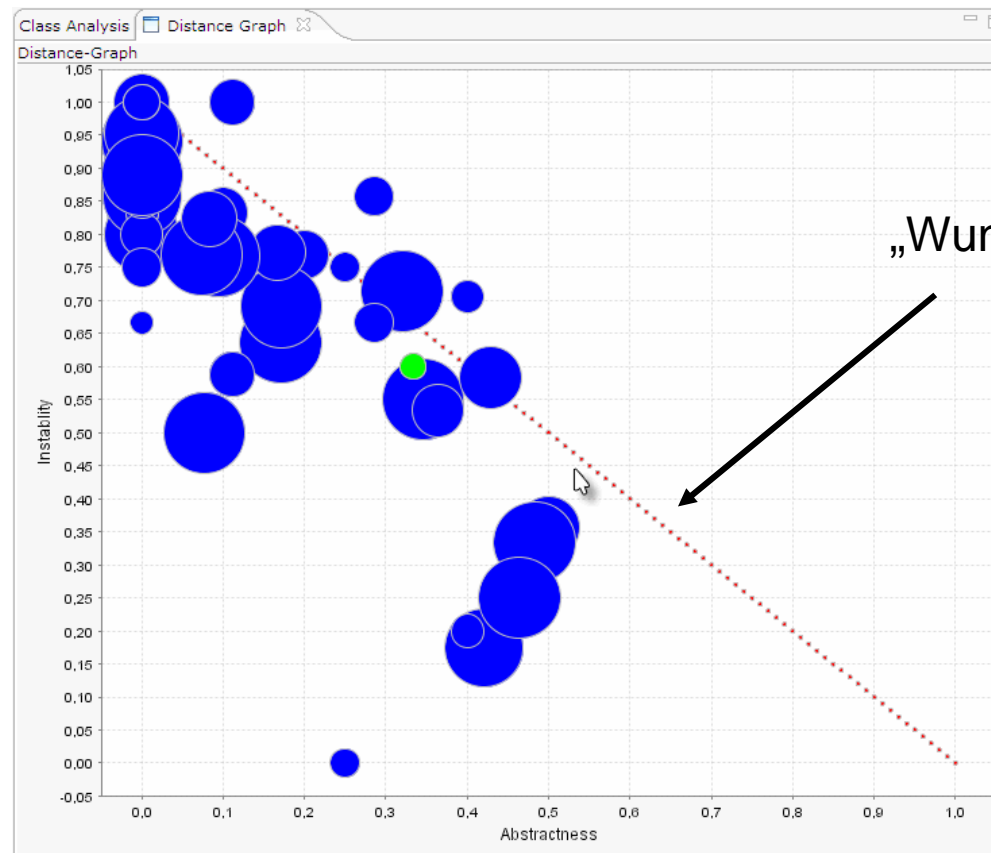
Klassen

Entfernung

Abhängigkeiten

<http://cap.xore.de/images/screenshots/overview.gif>

# Distance-Graph



<http://cap.xore.de/images/screenshots/distanceGraph.gif>

## Stil-Analyse

- n Leichtere Lesbarkeit
- n Schnelleres Einarbeiten in unbekanntem Code
- n Schnelleres Zurechtfinden in bekanntem Code
- n Teilweise bis zu 250 Zeichen pro Zeile(!)
- n Vorhandensein von JavaDoc / spezifischen Headern
- n Finden leerer Statements
- n Einhalten von Konventionen wie „Logger“ statt „println()“

## Beispiel: Checkstyle & PMD

- n Checkstyle

- n <http://checkstyle.sourceforge.net>

- n PMD

- n <http://pmd.sourceforge.net>

- n Version 3.2

- n Eclipse-Plugin

- n Download bei SourceForge

- n Pmd-eclipse

- n Version 3.0

- n Instruktionen: <http://pmd.sourceforge.net/eclipse/>

- n Update-Site: <http://pmd.sourceforge.net/eclipse/>



syngenio

We make IT work.

---

## PMD

- n Online „Warnungen“
- n Reports
  - n Html
  - n Excel
- n Regelsatz als XPath-Ausdrücke

## Programmfluss-Analyse

- n Weniger Fehler zur Laufzeit
- n Syntaxcheck durch Compiler und Editor
  - n Keine Kenntnis über Laufzeitverhalten
- n Nicht jeder Code(-Pfad) wird durch Tests abgedeckt
- n In großem Projekt auf Anhieb 100 mögliche NPE gefunden

## Programmfluss-Analyse (2)

- n Vom Compiler nicht angemerkert:

```
Object o = null;  
o.toString();
```



NPE

```
String foo = „Hello World“;  
foo.trim();
```



Rückgabe ignoriert

## Beispiel: Findbugs

- n Standalone – Werkzeug
  - n <http://findbugs.sourceforge.net>
  - n Version 0.9.1 – 15.6.2005
  - n Swing-GUI
  - n Komplette Analyse eines Projekts
- n Eclipse-Plugin
  - n Zusätzlicher Builder in Eclipse
  - n Direkte Markierung von Problemen
  - n Anzeige im „Problems“-View
  - n Leider aktuell noch FindBugs Version 0.8.8

## Fazit

- n Manche Ergebnisse sind falsch
- n Manche Checks sind zu viel

è Der Benutzer muss wissen was er tut

è Trotzdem große Hilfe

## More ...

### Open Source

- n [JPathFinder](#) - A verification VM written by NASA; supports a subset of the Java packages
- n [Jikes](#) - More than a compiler; now it reports code warnings too
- n [Lint4J](#) - Lock graph, DFA, and type analysis, many EJB checks
- n [Hammurapi](#) - Uses ANTLR, excellent documentation, lots of [rules](#)
- n [ESC/Java](#) - Finds null dereference errors, array bounds errors, type cast errors, and race conditions. Uses Java Modeling Language annotations.
- n [FindBugs](#) - works on bytecode, uses BCEL. Source code uses templates, nifty stuff!
- n [Jamit](#) - bytecode analyzer, nice [graphs](#)
- n [JWiz](#) - Research project, checks some neat stuff, like if you create a Button without adding an ActionListener to it. Neat.
- n [DoctorJ](#) - Written in C++. Uses Lex and YACC. Checks Javadoc, syntax and calculates metrics.
- n [JLint](#) - Written in C++. Uses data flow analysis and a lock graph to do lots of synchronization checks. Operates on class files, not source code.
- n [Checkstyle](#) - Very detailed, supports both Maven and Ant. Uses ANTLR.
- n [JCSC](#) - Does a variety of coding standard checks, uses JavaCC and the GNU Regexp package.

## More ...

### Commercial

- n [JStyle](#) - \$995, nice folks, lots of metrics and rules
- n [Energy Java Code Analyser](#) - 200 rules, lots of IDE plugins
- n [Simian](#) - fast, works with Java, C#, C, CPP, COBOL, JSP, HTML
- n [CodePro Studio](#) - \$589, works with Eclipse, lots of rules
- n [JTest](#) - Very nice with tons of features, but also very expensive and requires a running X server (or Xvfb) to run on Linux. They charge \$500 to move a license from one machine to another.
- n [Assent](#) - The usual stuff, seems pretty complete.
- n [AzoJavaChecker](#) - Rules aren't listed online so it's hard to tell what they have. Not sure how much it costs since I don't know German.
- n [Aubjex](#) - Rules aren't listed online. Appears to have some code modification stuff, which would be cool to have in PMD. \$299.
- n [Flaw Detector](#) - In beta, does control/data flow analysis to detect NullPointerExceptions
- n [AppPerfect](#) - 220 rules, produces PDF/Excel reports, std version is free, professional version \$500 and includes auto-fixing problems