Jürgen Nicolai
Geschäftsführender Gesellschafter

main {GRUPPE}

Liebknechtstrasse 33
70565 Stuttgart
Tel : 07 11/7 81 19 90
Fax : 07 11/ 7 81 19 91
Mail : j.nicolai@main-gruppe.de
Web: www.main-gruppe.de

# „Kampf dem Fehlerteufel“
# PMD, Findbugs und Checkstyle in großen Projekten


Niederlassung Leipzig
Reudnitzer Str. 13
04103 Leipzig
Tel : 03 41/ 9 98 20 06
Fax: 03 41/ 9 98 20 07

# Inhalt des Vortrages

- Statische Code-Analyse: grundlegende Konzepte
- PMD, FindBugs und CheckStyle
- Anpassung von PMD, FindBugs und Checkstyle
- Integration in Entwicklungsumgebungen
- Statische Code-Analyse während des Builds
- Analyse++: MAVEN Dashboard, XRadar & Co
- Die schönsten Fehler

# Statische Code-Analyse: grundlegende Konzepte

# Statische Code-Analyse: grundlegende Konzepte

„Statische Code-Analyse oder kurz statische Analyse ist ein statisches Software-Testverfahren.

Der Quelltext wird hierbei **einer Reihe formaler Prüfungen unterzogen**, bei denen bestimmte Sorten von Fehlern entdeckt werden können, noch **bevor die entsprechende Software ausgeführt wird“**

(Quelle: http://de.wikipedia.org/wiki/Statische_Code-Analyse).

→ Statische Code-Analyse soll **qualitative Schwachstellen im Source- Code** finden.

# Keine Compile-Fehler: Aber Was ist hier „falsch“?

```
import java.io.File;

public class TestJAX {

    public static Boolean FalsePositiveWarning(final String pParameter) {

        if (pParameter == "")
            return null;
        String s = "C:\\temp\\text.txt";
        pParameter.trim();
        if (s == pParameter) {
            File aFile = new File("C:\\temp\\test.txt");
            aFile.delete();
            return true;
        } else {
            return false;
        }

    }
    public static void main(final String[] args) {
        try {
            boolean returnValue = FalsePositiveWarning("");
            System.out.println(returnValue);
            returnValue = FalsePositiveWarning("   C:\\temp\\text.txt   ");
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
```

# ... ziemlich viel…

```
import java.io.File;

public class TestJAX {

    public static Boolean FalsePositiveWarning(final String pParameter) {

        if (pParameter == "")
            return null;
        String s = "C:\\temp\\text.txt";
        pParameter.trim();
        if (s == pParameter) {
            File aFile = new File("C:\\temp\\test.txt");
            aFile.delete();
            return true;
        } else {
            return false;
        }

    }
    public static void main(fir
        try {
            boolean returnValue = FalsePositiveWarning("");
            System.out.println(returnValue);
            returnValue = FalsePositiveWarning("    C:\\temp\
        } catch (Exception e) {
            e.printStackTrace();
```

System.out.print is used

Kein Kommentar?
ein Fehler ?

Funktion gibt null zurück.
Erwartet wird aber true oder false

Der Returnwert von trim() wird nicht zugewiesen

Der Return-Wert von delete() wird nicht geprüft

System.out.println
Wollen Sie das?

# ... oder ist das nur eine „Demo“?

- Aktuelle JBOSS Version 5.0.1
- Klasse org.jboss.mx.timer.JBossTimer

```
222
223     /**
224      * Gets a copy of the flag indicating whether a peridic notification is executed at fi
225      *
226      * @param id The timer notification identifier.
227      * @return A copy of the flag indicating whether a peridic notif
228      */
229     public Boolean getFixedRate(Integer id)
230     {
231        // Make sure there is a registration
232        RegisteredNotification rn = (RegisteredNotification) notifications.get(id);
233        if (rn == null)
234           return null; HEALTH4J >> : NP_BOOLEAN_RETURN_NULL
235
236        // Return a copy of the fixedRate
237        return new Boolean(rn.fixedRate); HEALTH4J >> : BooleanInstantiation
238     }
239
```

Funktion gibt u.U. null zurück. Erwartet wird aber true oder false

# Statische Code-Analyse: Bewertung


statische Analysen können automatisiert werden


geringe Initial-Aufwände


statische Analysen sind bereits in frühen Projektphasen möglich:

Es wird kein lauffähiges System benötigt


Keinerlei Aussagen über die Funktionalität der Software


Keine Aussagen über Performance


„Falschmeldungen“ müssen manuell geprüft werden


kein Ersatz für dynamische Verfahren z.B. Unit-Tests

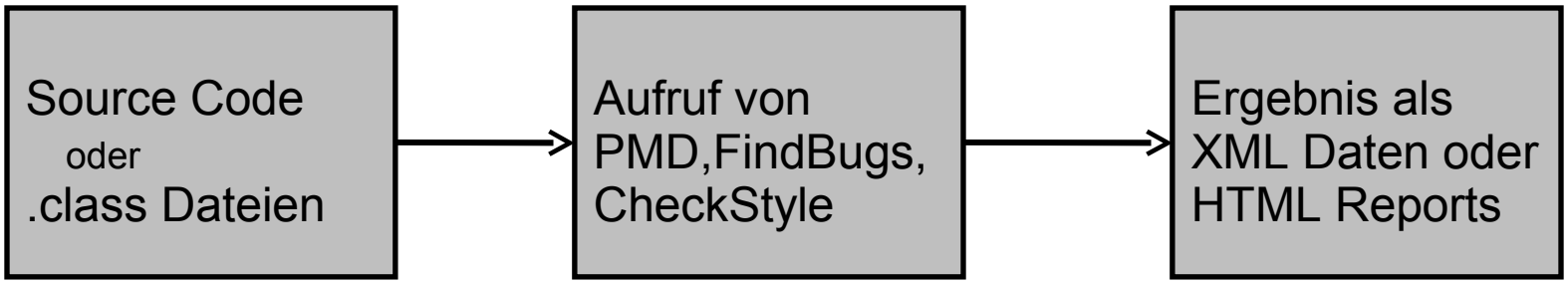
# PMD, FindBugs und CheckStyle

# OpenSource-Tools zur statische Code-Analyse

- PMD : http://pmd.sourceforge.net
- FindBugs : http://findbugs.sourceforge.net
- CheckStyle : http://checkstyle.sourceforge.net

# Statische Code-Analyse: Grundprinzipien

- Source Code oder compilierte .class Dateien werden analysiert
- Der Code kommt **nicht** zur Ausführung
- Einfache Ausführung der Tools als Batch Datei oder ANT Task
- Ausgabedaten sind XML oder HTML Dateien



# Statische Code-Analyse: Grundprinzipien

Analyse:

- HTML-Reports zum Teil „wenig sexy“ und unübersichtlich
- Sehr viele, wenig „relevante“ Fehler… >>>

## Summary

| Rule name | Number of violations |
|---|---|
| CollapsibleIfStatements | 159 |
| EmptyIfStmt | 62 |
| UnusedNullCheckInEquals | 1 |
| ForLoopShouldBeWhileLoop | 2 |
| UnnecessaryFinalModifier | 7 |
| ReturnFromFinallyBlock | 1 |
| UselessOverridingMethod | 6 |
| UnnecessaryReturn | 5 |
| UnconditionalIfStatement | 7 |
| EmptyWhileStmt | 1 |
| EmptyCatchBlock | 55 |
| BooleanInstantiation | 4 |
| ClassCastExceptionWithToArray | 3 |

## Detail

### PMD report

### Problems found

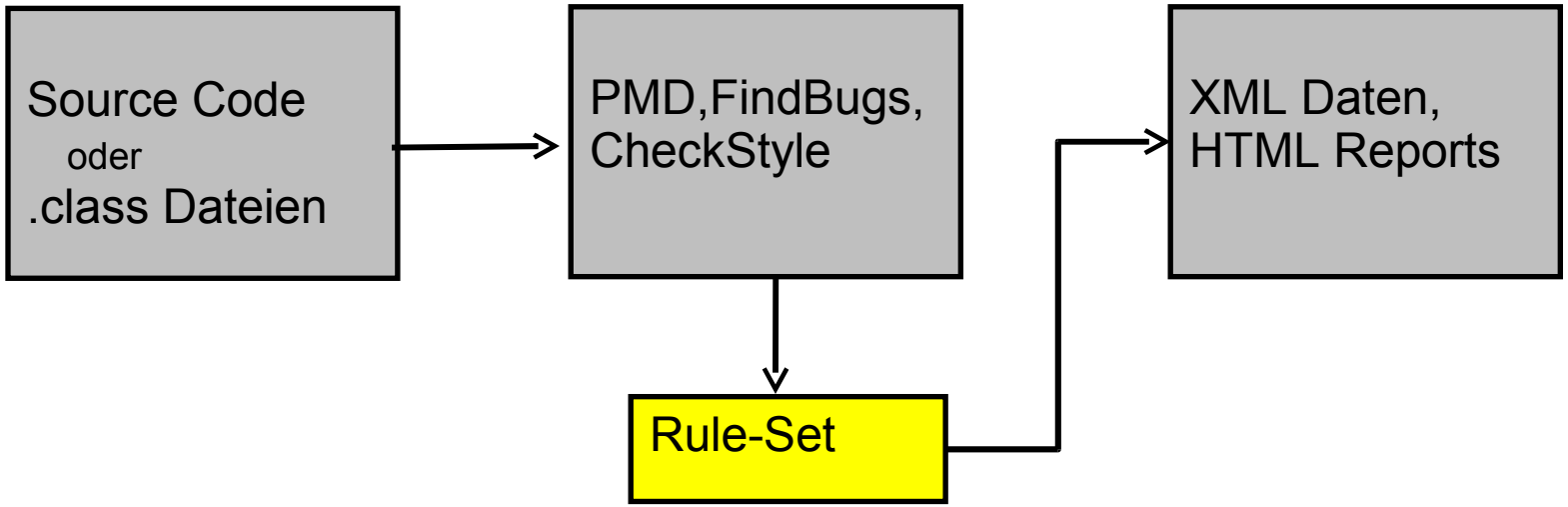
| # | File | Line | Problem |
|---|---|---|---|
| 1 | C:\argouml\src\argouml-app\src\org\argouml\application\events\ArgoEventPump.java | 377 | These nested if statements could be combined |
| 2 | C:\argouml\src\argouml-app\src\org\argouml\application\events\ArgoEventPump.java | 386 | These nested if statements could be combined |
| 3 | C:\argouml\src\argouml-app\src\org\argouml\application\events\ArgoEventPump.java | 394 | These nested if statements could be combined |

# Statische Code-Analyse: Rule-Set

Analyse:

- Jedes Werkzeug wendet eine bestimme Menge an Regeln an.
- Beispiel:

  „Jede Methode muss mit einem JavaDoc Kommentar versehen sein“
- Die Regeln eines Tools werden **„rule-set“** genannt
- PMD, FindBugs und CheckStyle prüfen etwa 700 Regeln >>>

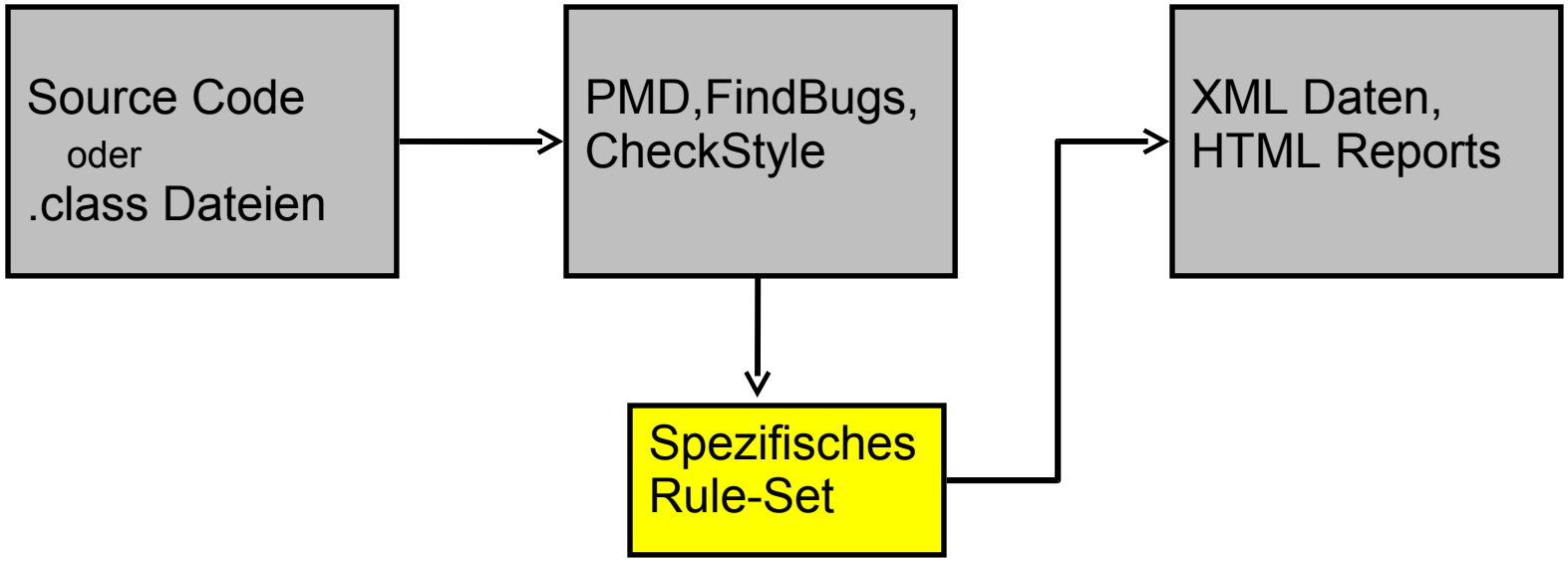


# Statische Code-Analyse: Rule-Set

Analyse:

- Die Tools zeigen in der Regel „zu viele“, „nicht relevante“ und „falsche“ Stellen an.
- Falsche Stellen = Fehlalarm= **„false positive warnings“**

→ Lösung: Das „rule-set“ wird Projekt-spezifisch angepasst, damit nur „relevante Fehler“ angezeigt werden



# Anpassung von PMD, FindBugs und CheckStyle

# Anpassung des rule-sets

```
import java.io.File;

public class TestJAX {

    public static Boolean FalsePositiveWarning(final String pParameter) {

        if (pParameter == "")
            return null;
        String s = "C:\\temp\\text.txt";
        pParameter.trim();
        if (s == pParameter) {
            File aFile = new File("C:\\temp\\test.txt");
            aFile.delete();
            return true;
        } else {
            return false;
        }

    }
    public static void main(final
        try {
            boolean returnValue = FalsePositiveWarning("");
            System.out.println(returnValue);
            returnValue = FalsePositiveWarning("    C:\\temp\
        } catch (Exception e) {
            e.printStackTrace();
```

System.out.print is used

Kein Kommentar? ein Fehler ?

Funktion gibt null zurück. Erwartet wird aber true oder false

Der Returnwert von trim() wird nicht zugewiesen

Der Return-Wert von delete() wird nicht geprüft

System.out.println Wollen Sie das?

# Anpassung des rule-sets

- Das rule-Set wird meist in Form von XML-Dateien definiert
- Es existiert leider kein einheitliches Verfahren, um ein rule-Set festzulegen

→ **Die Anpassung des rule-set ist aufwändig**

# Statische Code-Analyse: false positive warnings

Analyse:

- Falsche Stellen = **Fehlalarm**= **„false positive warnings“**
- „False positive warnings“ müssen erkannt und manuell aussortiert werden
- Beispiel: JDK 1.6, Klasse InetAdress:

```
/**
 * Compares this object against the specified object.
 * The result is <code>true</code> if and only if the argument is
 * not <code>null</code> and it represents the same IP address as
 * this object.
 * <p>
 * Two instances of <code>InetAddress</code> represent the same IP
 * address if the length of the byte arrays returned by
 * <code>getAddress</code> is the same for both, and eac
 * array components is the same for the byte arrays.
 *
 * @param   obj   the object to compare against.
 * @return  <code>true</code> if the objects are the sam
 *          <code>false</code> otherwise.
 * @see     java.net.InetAddress#getAddress()
 */
public boolean equals(Object obj) {
return false;
}
```

return false

Richtig
oder falsch?

## Statische Code-Analyse: false positive warnings

Analyse:

- „False positive warnings“ treten bei allen Tools auf
- Fehlalarme sind problematisch, da sie „von Hand“ analysiert und bearbeitet werden müssen
- Was tut man, wenn man Fehlalarme erkannt hat?

→ **Rule-Set anpassen**

→ **Fehlalarme im Source-Code kennzeichnen**

# Kennzeichnung Fehlalarme

- „Die Kennzeichnung von Fehlalarmen ist nicht standardisiert:
- FindBugs verwendet Annotations, PMD verwendet Kommentare und Annotations, CheckStyle verwendet XML Dateien

## Suppressing warnings

### Annotations

You can use a JDK 1.5 annotation to suppress PMD warnings, like this:

```
// This will suppress all the PMD warnings in this class
@SuppressWarnings("PMD")
public class Bar {
 void bar() {
  int foo;
 }
}
```

Meldungen werden unterdrückt

Or you can suppress one rule with an annotation like this:

```
// This will suppress UnusedLocalVariable warnings in this class
@SuppressWarnings("PMD.UnusedLocalVariable")
public class Bar {
 void bar() {
  int foo;
 }
}
```

PMD also obeys the JDK annotation @SuppressWarnings("unused"), which will apply to all rules in the unused ruleset

```
// This will suppress UnusedLocalVariable and UnusedPrivateMethod warnings in this class
@SuppressWarnings("unused")
```

# Tipps zur Anpassung des rule-Sets

- PMD, FindBugs und CheckSyle liefern so viele Hinweise, dass die Reporte nach 2 Wochen meist nicht mehr gelesen werden.
- **Das „rule-set“ muss deshalb unbedingt eingeschränkt werden.**
- Ausgangspunkt für das „rule-set“ können die „Programmierrichtlinien“ eines Projektes sein. Vorlage sind z.B. Sun Code Conventions: http://java.sun.com/docs/codeconv/
- Aktivierung von „harten Fehlern“ z.B. potentielle Nullpointer, falsche String-Vergleiche im „rule-set“.
- Deaktivieren /Anpassen von Regeln, die oft zu Fehlalarmen führen.

→ **Die Konfiguration ist aufwändig und erfordert fundiertes Java Know-how.**

# IDE Integration

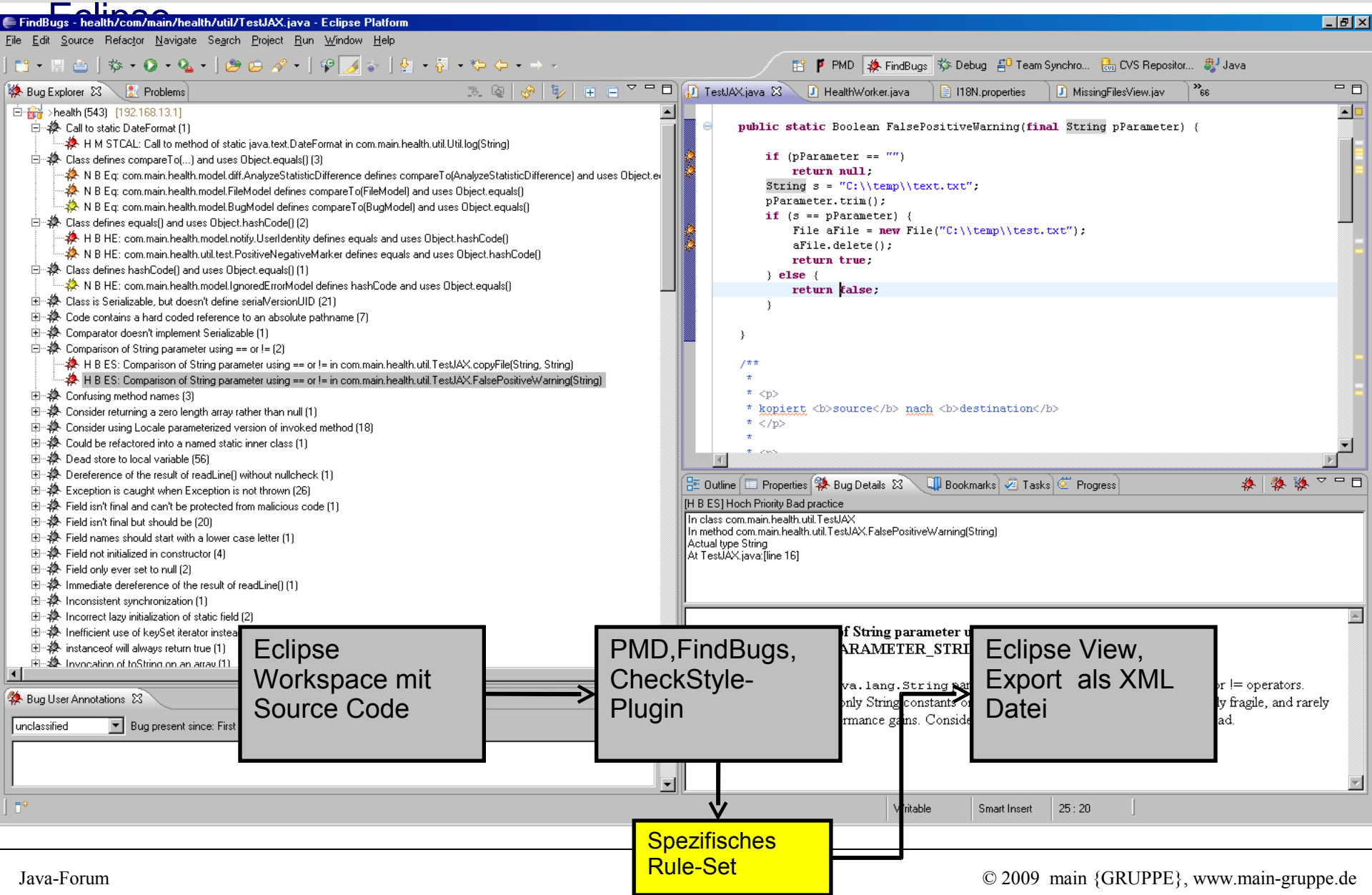
- PMD, FindBugs und CheckSyle können im Batch z.B. während des Builds aufgerufen werden.
- Wünschenswert wäre aber die Prüfung des Codes bereits in der Entwicklungsumgebung.
- Es existieren PMD, FindBugs und CheckStyle- Plugins für viele Entwicklungsumgebungen.

Eclipse Plugin CheckStyle:
http://eclipse-cs.sourceforge.net/

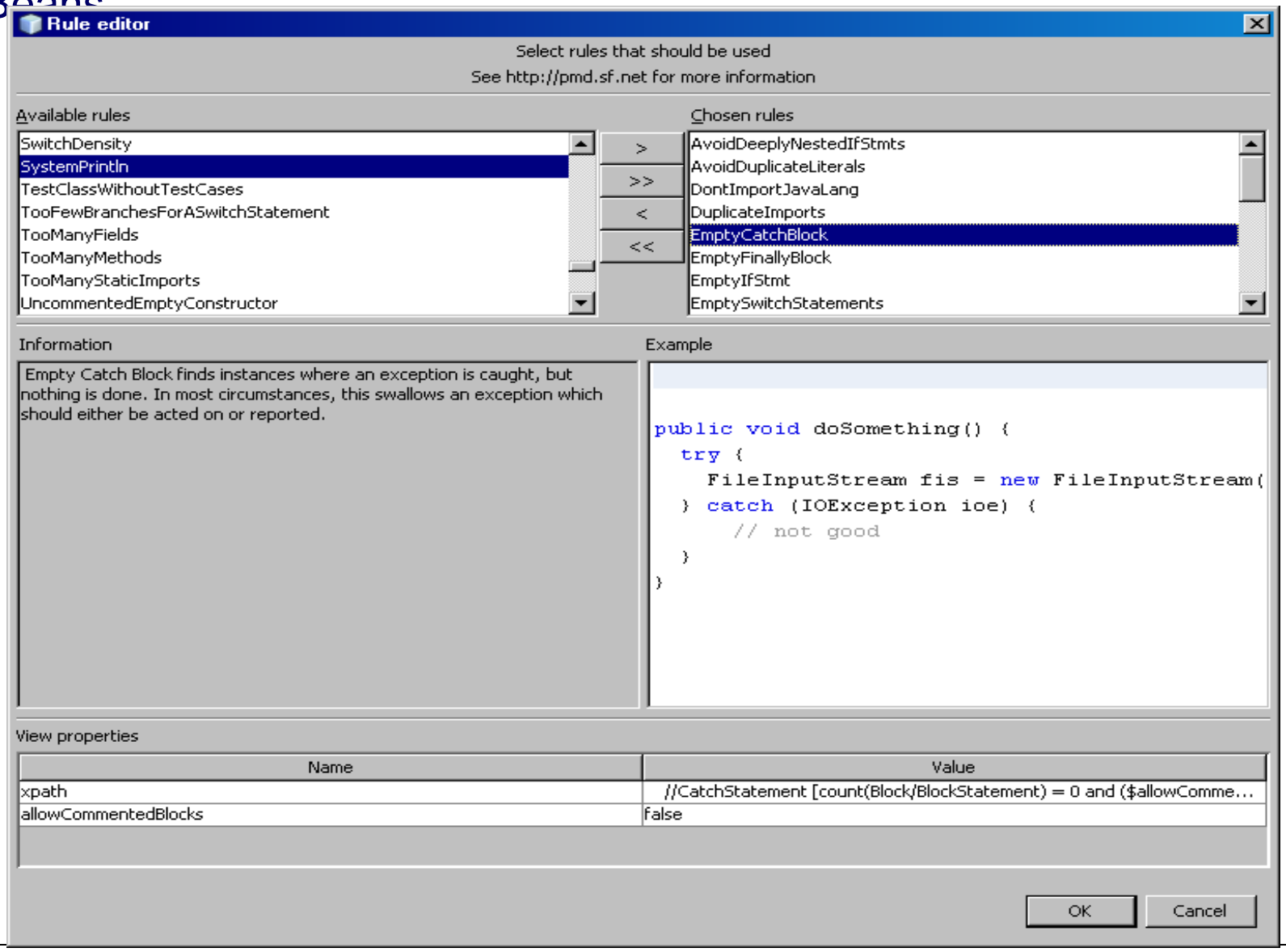
Eclipse Plugin FindBugs
http://findbugs.cs.umd.edu/eclipse

Eclipse Plugin PMD http://pmd.sourceforge.net/integrations.html#eclipse

# IDE Integration am Beispiel von Eclipse



# Anpassung „PMD- rule-sets“ in NetBeans



# Welches Tool ist das „beste Tool“?

## CheckStyle: 127 Regeln


Code Conventions, Naming, Code Duplikate,
JavaDoc -Kommentare, Metriken, eigene Regeln, Eclipse Plugin


CheckStyle findet keine „harten Bugs“ wie potentielle Nullpointer

## PMD: 246 Regeln,


Code Conventions, Naming, Code Duplikate, Metriken
potentielle Nullpointer, eigene Regeln, Eclipse Plugin

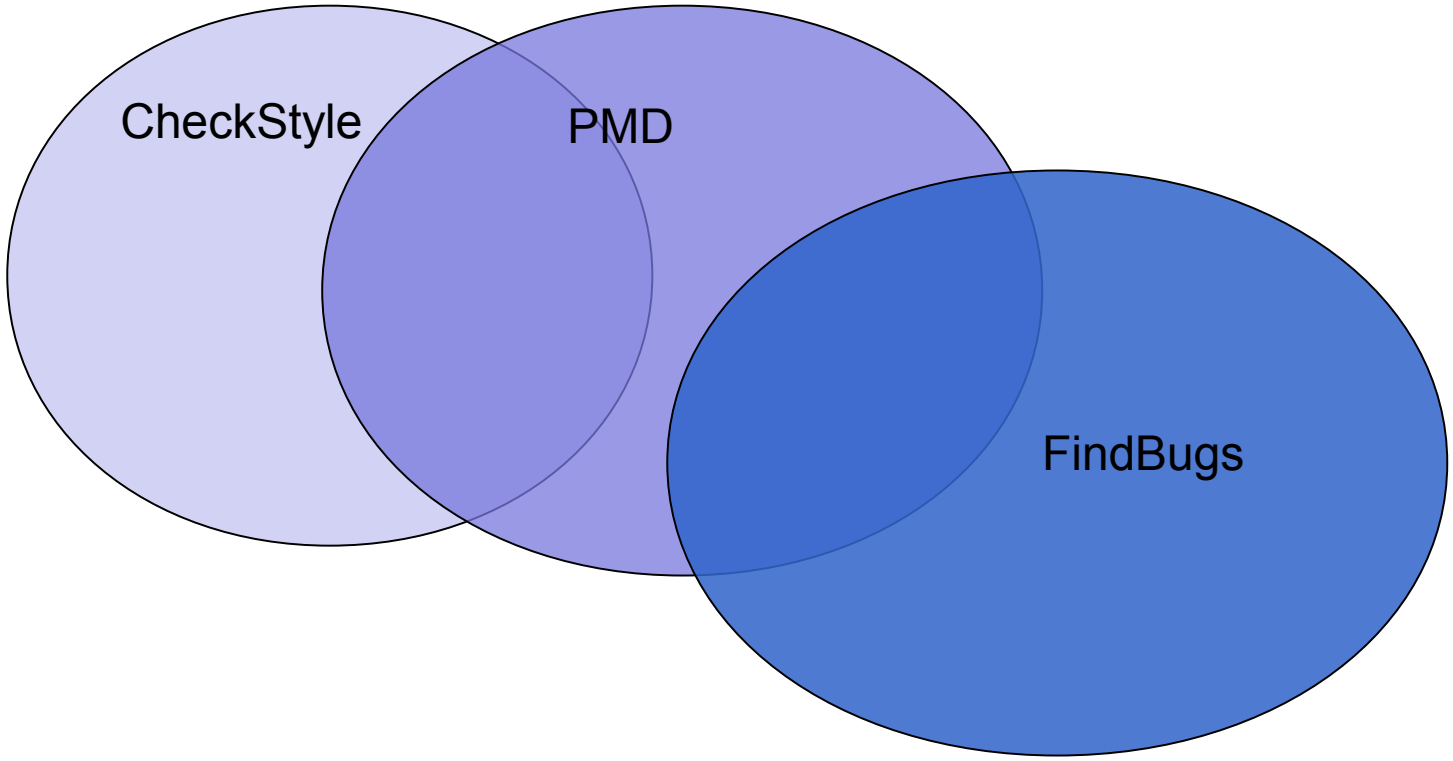

Keine JavaDoc Kommentare, weniger „harte Fehler“ als FindBugs

## FindBugs: 358 Regeln


„Harte Bugs“: Nullpointer, I/O Probleme u.v.a., Eclipse Plugin


Keine JavaDoc Kommentare, wenige Code-Conventions

# Welches Tool ist das „beste Tool“?




- Jedes Tool hat seine Stärken
- Die Regeln der Tools überdecken sich nur teilweise

**→ Für eine optimale Qualität werden oft mehrere Tools eingesetzt**

# Probleme bei Verwendung von mehreren Tools

- Jedes Tool generiert „eigene Reports“.
- Fehler werden von mehreren Tools gefunden.
- Man verliert den „Überblick“: Wo wurde welcher Fehler von welchem Tool gefunden?
- Jedes Tool hat „sein eigenes“ Eclipse Plugin mit eigenen Sichten auf die Fehler.
- Jedes Plugin wird anders bedient und konfiguriert.
- Entwickler führen die Analyse in der Entwicklungsumgebung oft nicht durch (z.B. aus Zeitgründen).

# Probleme bei Verwendung von mehreren Tools

- „Rule-Set“ muss für jedes Tool angepasst werden.
- Jedes Tool generiert „eigene Reports“.
- Keine einheitliche Kennzeichnung von Falschmeldungen, Beispiel:

```
@edu.umd.cs.findbugs.annotations.SuppressWarnings
                    (value="NM_METHOD_NAMING_CONVENTION")

@SuppressWarnings("PMD.MethodNamingConventions")

public void DoSomething() {

}
```

FindBugs und PMD Meldungen werden unterdrückt

**→ Code-Analyse während des Builds ist weiterhin notwendig.**

**→ Integration der Werkzeuge wird notwendig.**

# Integration von Entwicklungsumgebungen

# Integration von PMD, FindBugs und CheckStyle

- Für den Build von Java Systemen wird oft **ANT** (ant.apache.org) oder **MAVEN** (maven.apache.org ) eingesetzt
- FindBugs, CheckStyle und PMD lassen sich in ANT über in den Tools enthaltene „ANT-Tasks“ einfach integrieren
- Die HTML Reports sind aber wenig ansprechend und nicht mit dem Source Code verlinkt

**CheckStyle Audit**

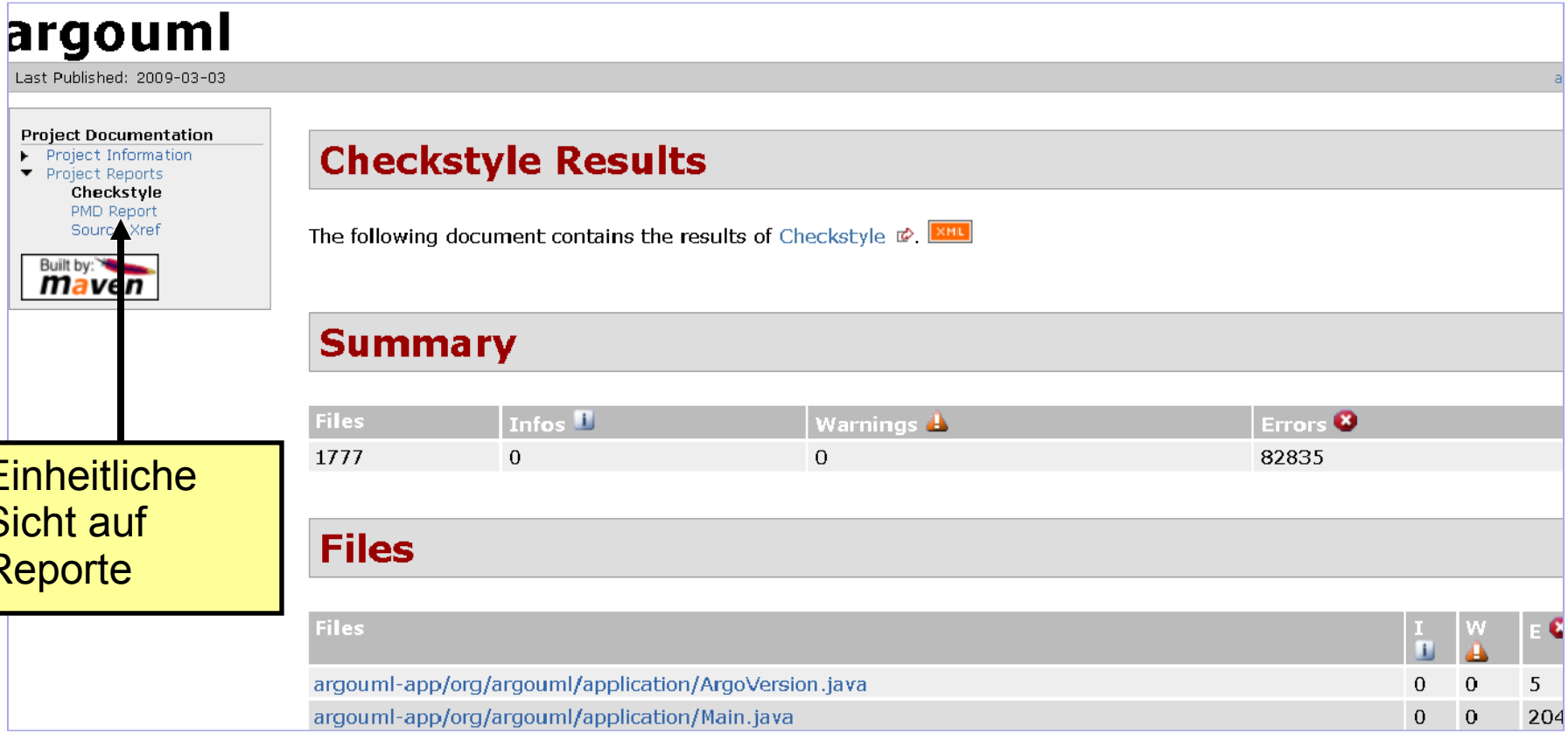
Designed for use with CheckStyle and Ant.

| Summary | |
|---|---|
| **Files** | **Errors** |
| 1777 | 82705 |

| Files | |
|---|---|
| **Name** | **Errors** |
| C:\argouml\src\argouml-app\src\org\argouml\uml\reveng\java\JavaRecognizer.java | 6798 |
| C:\argouml\src\argouml-app\src\org\argouml\uml\reveng\java\JavaLexer.java | 5566 |
| C:\argouml\src\argouml-app\src\org\argouml\language\java\generator\JavaRecognizer.java | 4142 |
| C:\argouml\src\argouml-app\src\org\argouml\language\java\generator\JavaLexer.java | 2307 |

# Integration von PMD, FindBugs und CheckStyle

- Bei der MAVEN Integration werden die HTML Reporte in einem einheitlichen „Look&Feel“ erzeugt.
- Fehler werden mit dem Source-Code verlinkt.




Einheitliche Sicht auf Reporte

# Wunschkonzert ..

.

- Auch bei  MAVEN werden die HTML Reporte „getrennt“ angezeigt.
- Man muss sich „mühsam durchklicken“, um neue Fehler zu finden.

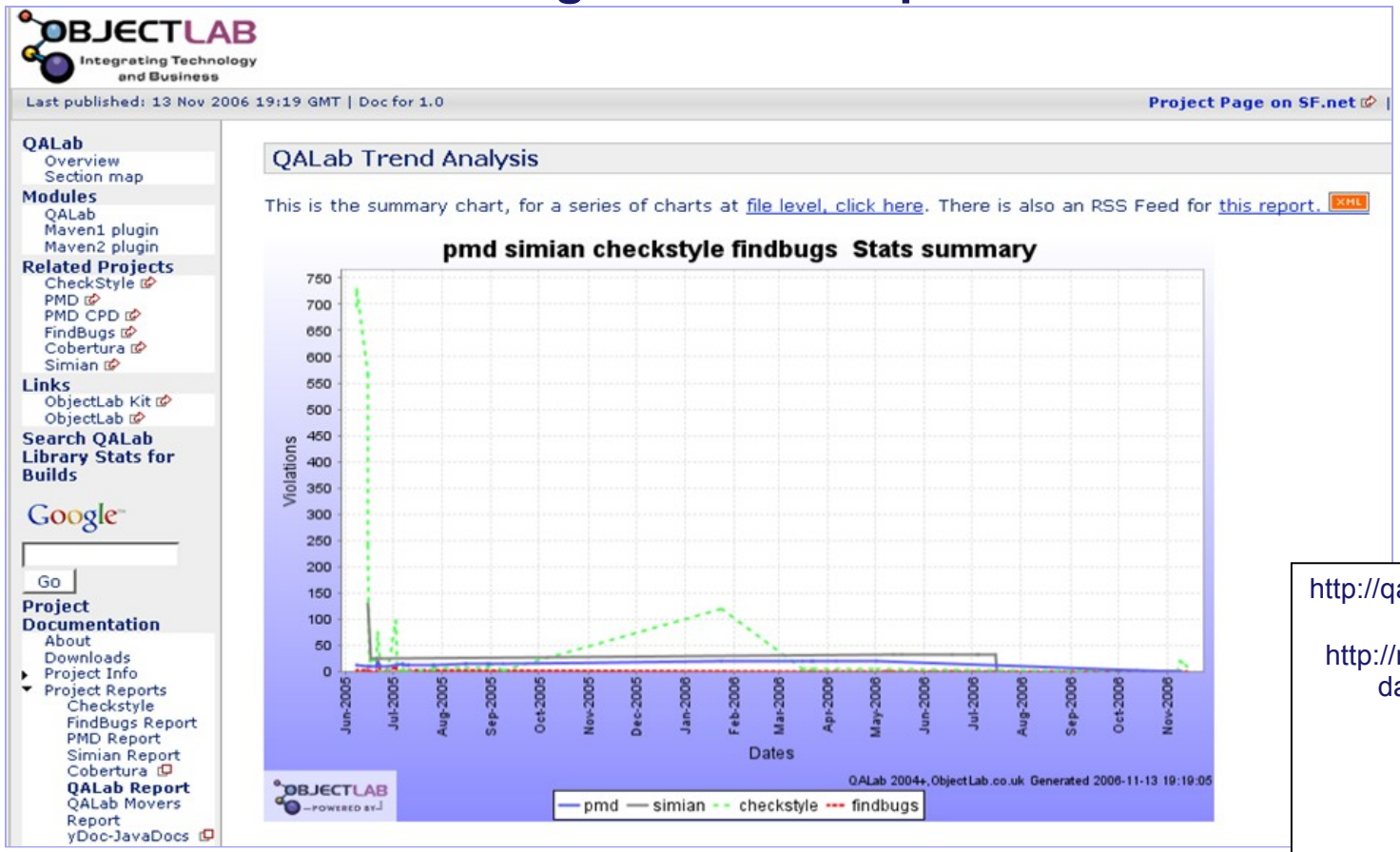
**Was „fehlt“ für die Verwendung in großen Projekten?**

- Ein „Meta-Plugin“ für die Entwicklungsumgebungen, welches die Fehler von CheckStyle, PMD und FindBugs in einer Sicht zeigt
- Bessere Integration der Tools ( z.B. einheitliches rule-set)
- Einheitliche Sicht auf die Gesamtqualität (ein Dashboard)
- Historisierung: „Wie viele Fehler wurden letzte Woche gefixt?“

→ **Lösungen für diese Wünsche gibt es nur teilweise**

# QALab, MAVEN-Dashboard Integration

- QALab oder MAVEN-Dashboard integrieren PMD, CheckStyle und FindBugs zu einer einheitlichen Plattform
- Die Tools bieten **Integration der Reporte und Historisierung**



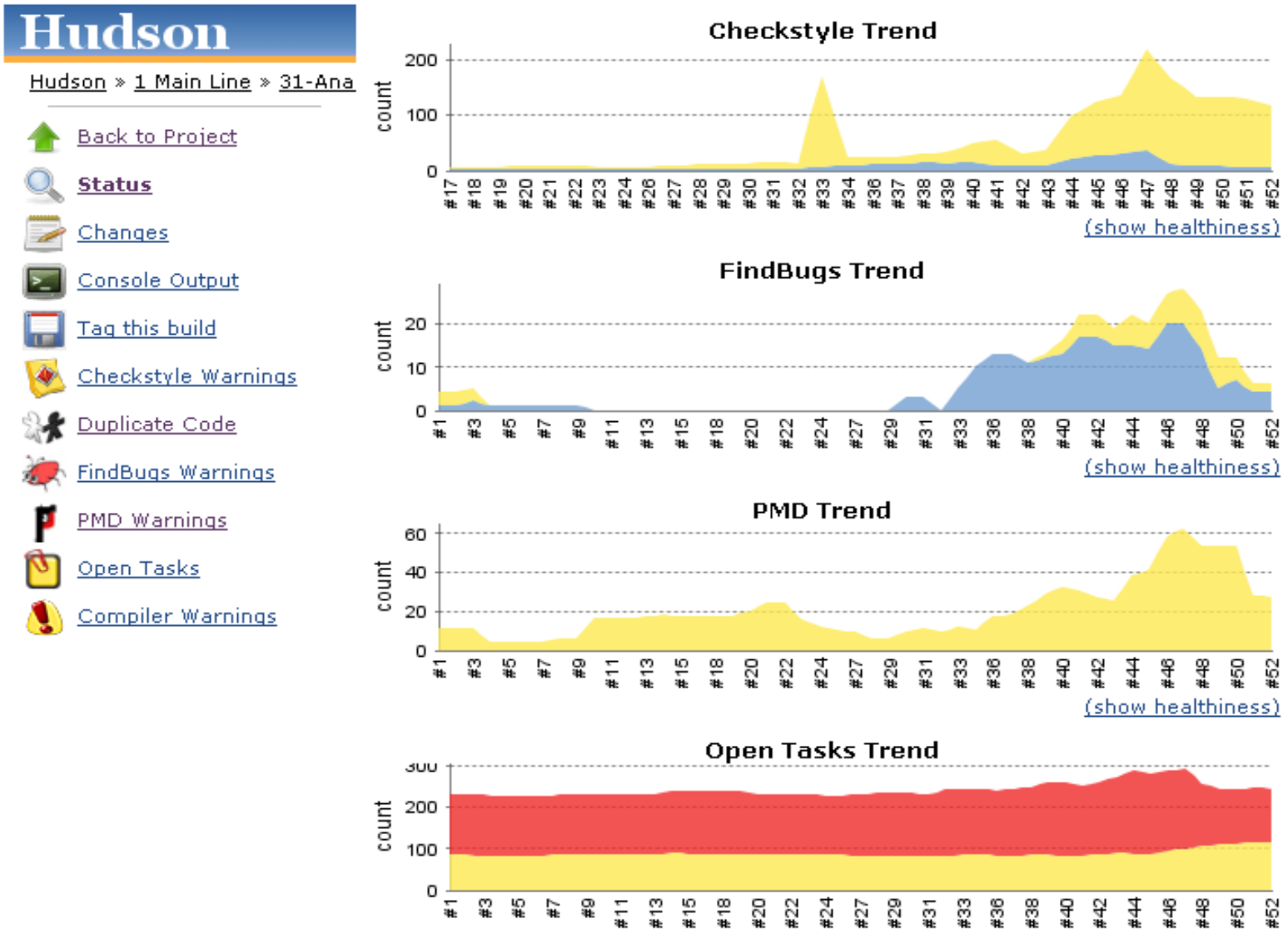

http://qalab.sourceforge.net

http://mojo.codehaus.org/dashboard-maven-plugin/

# Hudson Integration

(https://hudson.dev.java.net)

a bit further up on the same page.



# Grenzen der statischen Code-Analyse

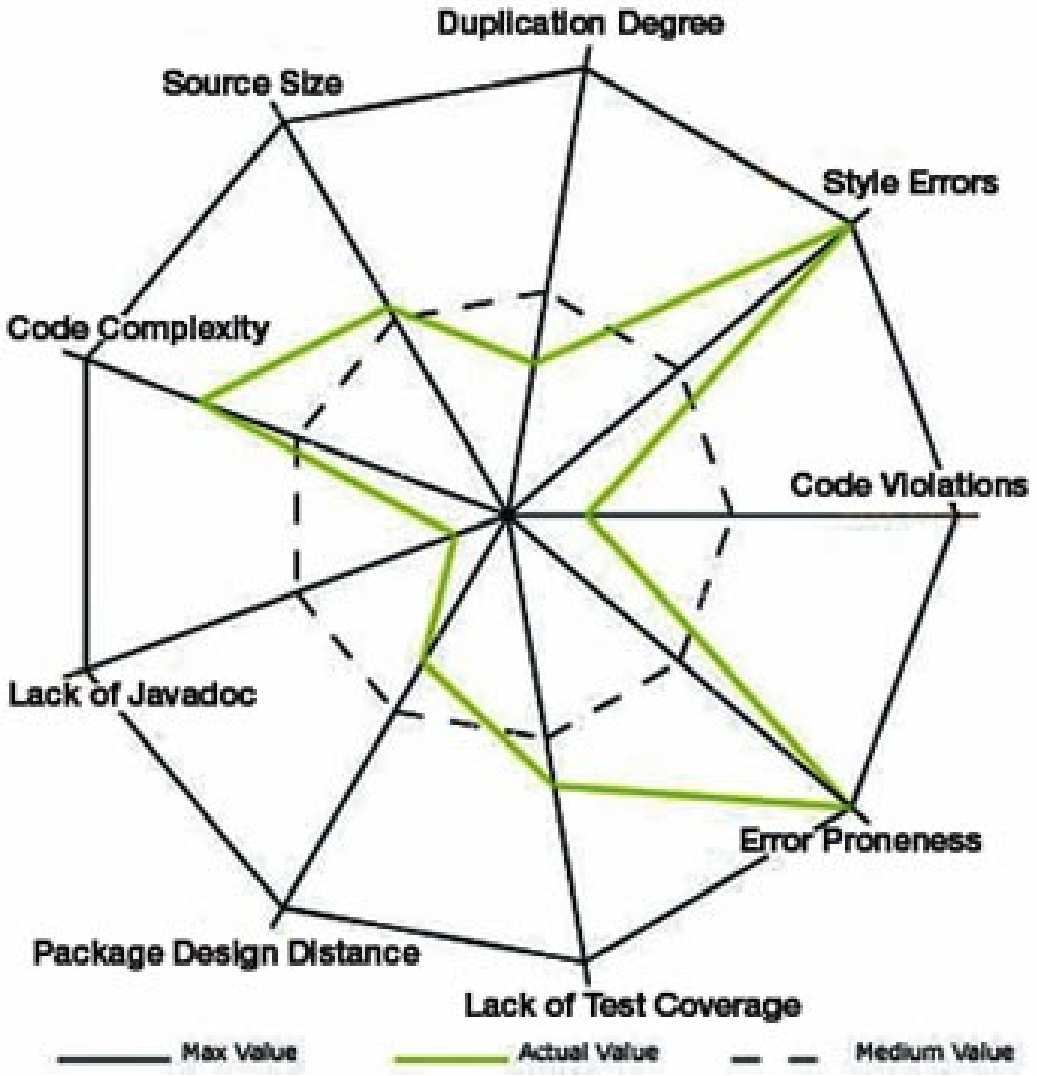
- PMD, CheckStyle und FindBugs untersuchen den Source-Code auf Fehler.
- Schlechtes Design z.B. zyklische Abhängigkeiten zwischen Klassen werden nicht untersucht.
- Die „Qualität der Architektur“ wird nicht beurteilt.

→ **PMD, FindBugs und CheckStyle führen kein Architekturmanagement durch.**

- Für das Architekturmanagement gibt es andere Produkte z.B. JDepend, ClassCycle, XRadar (www.xradar.sourceforge.net).

# Grenzen der statischen Code-Analyse

- Beim Architekturmanagement werden weitere Faktoren berücksichtigt z.B. Abhängigkeiten zwischen Paketen



# Die schönsten Fehler ...

# Die schönsten Fehler ...

- Unter www.japhara.de werden ca. 1,5 Mio Lines Of Code mit PMD, FindBugs und CheckStyle analysiert.
- Analysiert wurden z.B. das JDK, JBOSS, Spring und Tomcat.
- Es wurde ein in Projekten bewährtes „rule-set“ verwendet.
- Man erkennt sehr schnell, welche Projekte statische Codeanalyse konsequent nutzen.

# Die schönsten Fehler

| | | |
|---|---|---|
| **1** | **Ressourcen werden nicht freigegeben** | **Ressourcen werden nicht korrekt freigegeben z.B. kein close() von Streams oder Datenbank-Verbindungen. Oft werden Ressourcen nicht freigegeben, wenn eine Exception auftritt** |
| 2 | Objekte werden mit == statt mit equals() verglichen | If (aString == bString)<br>{<br><br>} |
| 3 | Eine Funktion gibt null anstatt true oder false zurück | Boolean f()<br>{<br><br>return null;<br>} |

# Die schönsten Fehler ...

- JDK1.6, Klasse org.apache.xerces.internal.impl.xs.XMLSchemaLoader

```
818             return saxToXMLInputSource((InputSource) val);
819         } else if (val instanceof InputStream) {
820             return new XMLInputSource(null, null, null,
821                     (InputStream) val, null);
822         } else if (val instanceof File) {
823             File file = (File) val;
824             InputStream is = null;
825             try {
826                 is = new BufferedInputStream(new FileInputStream(file));
827             } catch (FileNotFoundException ex) {
828                 fErrorReporter.reportError(XSMessageFormatter.SCHEMA_DOMAIN,
829                         "schema_reference.4", new Object[] { file.toString() },
830                         XMLErrorReporter.SEVERITY_ERROR);
831             }
832             return new XMLInputSource(null, null, null, is, null);
833         }
834         throw new XMLConfigurationException(
835                 XMLConfigurationException.NOT_SUPPORTED, "\""+JAXP_SCHEMA_SOURCE+
836                 "\" property cannot have a value of type {"+val.getClass().getName()+
837                 "}. Possible types of the value supported are String, File, InputStream, "+
838         "InputSource OR an array of these types.");
839     }
840
```

Wird dieser Stream korrekt geschlossen?

# Die schönsten Fehler ...

| | | |
|---|---|---|
| **4** | **Clone() return null** | **Die clone()-Methode kann unter gewissen Umständen den unerlaubten Rückgabewert null liefern.** |
| 5 | Falsche if- Abfragen | If ( maxH <1 \|\| maxH < 1)<br>{<br><br>} |
| 6 | Clone() wird falsch implementiert | Die Methode clone() wurde implementiert ohne dass von der Klasse das Interface Clonable implementiert wird |

# Die schönsten Fehler ...

- JDK1.6, Klasse java.swing.plaf.nimbus.AbstractRegionPainter

```
/**
 * Creates a new PaintContext.
 *
 * @param insets The stretching insets. May be null. If null, then assumed to be 0,
 * @param canvasSize The size of the canvas used when encoding the various x/y valu
 *                   If null, then it is assumed that there are no encoded values,
 *                   to one of the "decode" methods will return the passed in value
 * @param inverted Whether to "invert" the meaning of the 9-square grid and stretch
 * @param cacheMode A hint as to which caching mode to use. If null, then set to no
 * @param maxH The maximium scale in the horizontal direction to use before punting and redrawing from scratch.
 *             For example, if maxH is 2, then we will attempt to scale any cached images up to 2x the canvas
 *             width before redrawing from scratch. Reasonable maxH values may improve painting performance.
 *             If set too high, then you may get poor looking graphics at higher zoom levels. Must be >= 1.
 * @param maxV The maximium scale in the vertical direction to use before punting and redrawing from scratch.
 *             For example, if maxV is 2, then we will attempt to scale any cached images up to 2x the canvas
 *             height before redrawing from scratch. Reasonable maxV values may improve painting performance.
 *             If set too high, then you may get poor looking graphics at higher zoom levels. Must be >= 1.
 */
public PaintContext(Insets insets, Dimension canvasSize, boolean inverted,
                    CacheMode cacheMode, double maxH, double maxV) {
        if (maxH < 1 || maxH < 1) {  HEALTH4J >>  :  RpC REPEATED CONDITIONAL TEST
            throw new IllegalArgumentException("Both maxH and maxV must be >= 1");
        }

        this.stretchingInsets = insets == null ? EMPTY_INSETS : insets;
        this.canvasSize = canvasSize;
```

Das muss wohl maxV heißen…

# Fazit

- Statische Code-Analyse ist ein wichtiger Baustein bei der Sicherung der Code-Qualität
- Statische Code-Analyse kann mit wenig Aufwand betrieben werden
- Die Regeln von FindBugs, PMD und CheckStyle müssen angepasst werden.
- Die Tools sollten, sowohl in der Entwicklungsumgebung, als auch während des Builds eingesetzt werden.
- Zusätzliche Tools, wie QALab oder XRadar sind sinnvoll.
- Die schönsten Reports nutzen nichts, wenn sie nicht beachtet werden….

# Fragen ?

# Vielen Dank für Ihre Aufmerksamkeit!