## Abgabe in EiSE – Übungszettel 5 – WiSe 2015/16

Von Adam Shafei, Len Williamson & Steffen Pegenau

## Tools zur Softwarequalitätssicherung

### FindBugs

<http://findbugs.sourceforge.net/downloads.html>

#### Beschreibung:

* + - 1. FindBugs inspiziert den Java Byte Code und versucht mit statischen Analysemethoden Fehlermuster („Bug Patterns“) zu finden, die vorher hinterlegt sein müssen.

#### Unpräzise Ergebnisse:

#### Probleme :

1. Problem  
   Beschreibung:  
   Mögliche Lösung:
2. Problem  
   Beschreibung:  
   Mögliche Lösung:
3. Problem  
   Beschreibung:  
   Mögliche Lösung:

### PMD

<https://pmd.github.io/>

#### Beschreibung:

* + - 1. PMD untersucht den Quellcode anhand von Regelwerken auf typische Programmierfehler.

#### Unpräzise Ergebnisse:

#### Probleme :

1. Problem  
   Beschreibung:  
   Mögliche Lösung:
2. Problem  
   Beschreibung:  
   Mögliche Lösung:
3. Problem  
   Beschreibung:  
   Mögliche Lösung:

### CheckStyle

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

#### Beschreibung:

* + - 1. CheckStyle prüft Quellcode auf die Einhaltung von (selbstgesetzten) Programmierregeln.

#### Unpräzise Ergebnisse:

#### Probleme :

1. Problem  
   Beschreibung:  
   Mögliche Lösung:
2. Problem  
   Beschreibung:  
   Mögliche Lösung:
3. Problem  
   Beschreibung:  
   Mögliche Lösung:

### JDepend

<http://clarkware.com/software/JDepend.html>

#### Beschreibung:

* + - 1. JDepend generiert nach einer Quelltextanalyse Metriken zur Messung der Design-Qualität (Erweiterbarkeit, Wiederverwendbarkeit, Wartbarkeit, Abhängigkeiten).

#### Unpräzise Ergebnisse:

#### Probleme :

1. Problem   
   Beschreibung:  
   Mögliche Lösung:
2. Problem  
   Beschreibung:  
   Mögliche Lösung:
3. Problem  
   Beschreibung:  
   Mögliche Lösung:

### Dependency Finder

<http://depfind.sourceforge.net/>

#### Beschreibung:

* + - 1. DependencyFinder erzeugt aus Basis einer Bytecode-Analyse Abhängigkeitsgraphen und Qualitätsmetriken (z.B. Codezeilen / Methode)

#### Unpräzise Ergebnisse:

* + - 1. Da DependencyFinder nur Metriken und Abhängigkeitsgraphen erzeugt, liegt die Gefahr vor allem in einer unpräzisen Interpretation durch den Anwender. Grenzwerte müssen für die einzelnen Aspekte sinnvoll angepasst werden, um die Interpretation zu erleichtern.

### Checker Framework

<http://types.cs.washington.edu/checker-framework/>

#### Beschreibung:

* + - 1. Das Checker Framework erweitert das Typsystem von Java mit Annotationen, um dem Nutzer das Finden und Vermeiden von Fehlern zu erleichtern.

#### Unpräzise Ergebnisse:

* + - 1. Das Checker Framework ist auf Annotationen angewiesen. Ohne diese Annotationen die dem Framework Zusatzinformationen zum Code liefern, bleiben die Ergebnisse unpräzise. Durch das Hinzufügen von Annotationen schon beim Programmieren, lässt sich die Zahl der fehlerhaften Treffer reduzieren.

## Drei Funde des Checker Frameworks

### 1. Fund: Nullness Checker

Description Resource Path Location Type incompatible types in argument.

ftp = new FTPHTTPClient(proxyHost, proxyPort, proxyUser, proxyPassword);

found : @Initialized @Nullable String

required: @Initialized @NonNull String FTPClientExample.java /commons-net/src/main/java/examples/ftp line 217 Checker Framework Problem

Wir befinden uns in der main-Methode einer Klasse. Die beiden Parameter „proxyUser“ und „proxyPassword“ wurden dort zunächst mit „null“ initialisiert. Nur wenn bestimmte Aufrufparameter des Programms gesetzt sind, wird jeweils das „null“ überschrieben. Der Konstruktor speichert die beiden Angaben ohne weitere Prüfung in Attributen. Wird auf diese Attribute ohne weitere Prüfung zugegriffen, kann es zu Problemen kommen.

### 2. Fund: Linear Checker

Description Resource Path Location Type invalid type in catch argument.

} catch (Exception e) {

found : @Normal

required: @Unusable DefaultBeanIntrospector.java /commons-beanutils/src/main/java/org/apache/commons/beanutils line 141 Checker Framework Problem

Der „Linear Checker“ soll sicherstellen, dass es nie mehr als eine nutzbare Referenz auf ein Objekt gibt. Die Warnung wird hier gegeben, weil „e“ im Catch-Block als Referenz weiter gegeben wird:  
log.error("Error setting indexed property read method", e);

### 3. Fund: Tainting Checker

Description Resource Path Location Type

@TAINTED Object remove(@TAINTED DelegateFastHashMap this, @TAINTED Object p0) in org.apache.commons.beanutils.locale.LocaleConvertUtilsBean.DelegateFastHashMap cannot override V extends @TAINTED Object remove(@TAINTED AbstractMap<K extends @TAINTED Object, V extends @TAINTED Object> this, @TAINTED Object p0) in java.util.AbstractMap; attempting to use an incompatible return type

public Object remove(Object key) {

found : @TAINTED Object

required: V extends @TAINTED Object LocaleConvertUtilsBean.java /commons-beanutils/src/main/java/org/apache/commons/beanutils/locale line 568 Checker Framework Problem

Der „Tainting Checker“ soll sicherstellen, dass Daten aus unsicheren Quellen, beispielsweise Benutzereingaben, nicht ungeprüft verarbeitet werden. Hier entsteht die Warnung, weil in Annotationen nicht darauf eingegangen wurde, dass auch eine Kind-Klasse zurückgegeben werden kann.