Softwarequalitätssicherung

# Statische Analyse & Code-Reviews


**Gruppenmitglieder:** Dennis Jongebloed
Julian Hinxlage
Johannes Theiner

**Semester:** Sommersemester 2020
**letzte Änderung:** 14. Mai 2020

# 1 Statische Analyse - Projekt

PMD verwendet Codacy, SonarQube, Checkstyle und auch PMD selbst, als statische Analyse Tools um den eigenen Code zu analysieren. Die Tools sind im CI Server konfiguriert und werden bei jedem Commit neu ausgeführt und die Metriken aktualisiert. Die Metriken sind also immer aktuell. Zum aktuellen Stand hat SonarQube folgende Ergebnisse:

- 41 Bugs
- 50 Vunlnerabilities
- 32 Security Hotspots
- 6400 Code Smells
- 173 Tage Technical Debt
- 185 Duplicated Blocks
- 6,3% Duplications

Und Codacy hat folgende Ergebnisse:

- 4446 total issues
- 2385 Error Prone
- 1593 Code Style
- 468 Performance

Codacy und SonarQube stellen online Dashboards zur Verfügung:
Codacy Dashboard: https://app.codacy.com/manual/pmd/pmd/dashboard
SonarQube Dashboard: https://sonarcloud.io/dashboard?id=net.sourceforge.pmd:pmd

Checkstyle hat als Ergebniss 0 violations. Da Checkstyle nur Code Formatierungen testet, werden diese vermutlich automatisch behoben(Code Reformat). Wodurch die 0 violations entstehen.

Die verwendeten Mektriken sind aus unserer Sicht angemessen. Man kann schnell sehen wie viele Fehler vorhanden sind und an welcher stelle im Code diese auftreten. Neben den Metriken kann man sich auch alle Fehler einzelned angucken und diese Fixen.

# 2 Code-Reviews Beurteilung

## 2.1 Walkthrough

Vorteile:

- Geringer Aufwand, da Vor- und Nachbereitung von gerigem Umfang bzw. nicht zwingend erforderlich sind.
- Für kleine Entwicklungsteams von bis zu 5 Personen geeignet
- Sinnvoll für unkritischen Dokumenten
- Da Benutzungssituationen einbezogen werden können, können zudem Unvollständigkeiten und Missverständnisse identifiziert werden.

Nachteile:

- Die Überarbeitung des zu prüfenden Objekts wird vom Autor entschieden. Dies wird nicht nachgeprüft.
- Wenige Fehler bzw. Defekte werden identifiziert.
- Da der Autor die Sitzung leitet, kann er diese auch zu sehr dominieren.

Aufgrund der niedrigen Anzahl identifizierten Fehler und Defekte, ist diese Reviewart weniger für PMD geeignet.

## 2.2 Inspektion

Vorteile:

- Prüfschritte mit Eintritts- und Austrittskriterien definiert.
- Aufdeckung von Unklarheiten, möglichen Fehlern und Defekten, Bestimmung der Qualität des Prüfobjekts, Verbesserung der Qualität des Prüf- und Entwicklungsprozess.
- Da eine begrenzte Anzahl von Aspekten bei einer Inspektion behandelt wird, können sich die Gutachter spezifischer vorbereiten. Somit können Fehler und Defekte besser erkannt werden.
- Vor der Inspektion wird das Prüfobjekt formal auf reiewfähigkeit geprüft.

Nachteile:

- Die beteiligten Personen sind meist direkte Kollegen, keine Außenstehenden Personen.
- Wenige Fehler bzw. Defekte werden identifiziert, da eher z.B. auf die Nichteinhaltung von Entwicklungsstandard geachtet wird.

Die Gutachter haben sich auf eine begrenzte Anzahl von Aspekten spezifisch vorbereitet, so können Fehler und Defekte gut identifiziert werden. Da PMD am besten keine Fehler oder Defekte besitzen darf, ist diese Reviewart sehr gut geeignet.

## 2.3 Technisches Review

Vorteile:

- Gutachter sind Fachexperten.
- Genaue Überprüfung des Prüfobjekts (Spezifikation, Standards, Einsatzbereit).
- Individuelle Prüfung anhand der Prüfkriterien durch Gutachter.
- Es können viele Fehler und Defekte identifiziert werden, da die Gutachter Fachexperten sind.

Nachteile:

- Hoher Aufwand in der Vorbereitungsphase.
- Das Managements entscheidet über die Konsequenzen des Ergebnisses.

Da die Gutachter Fachexperten sind, können auch besser Fehler und Defekte identifiziert werden. Der hohe Aufwand in der Vorbereitungsphase ist für ein Projekt wie PMD kein Ausschlusskriterium, da im Gegenzug viele Fehler entdeckt werden können.

## 2.4 Informeller Review

Vorteile:

- Einfache Form der grundlegenden Vorgehensweise (abgeschwächte Form des Reviews). Geringer Aufwand.
- Geringe Kosten.

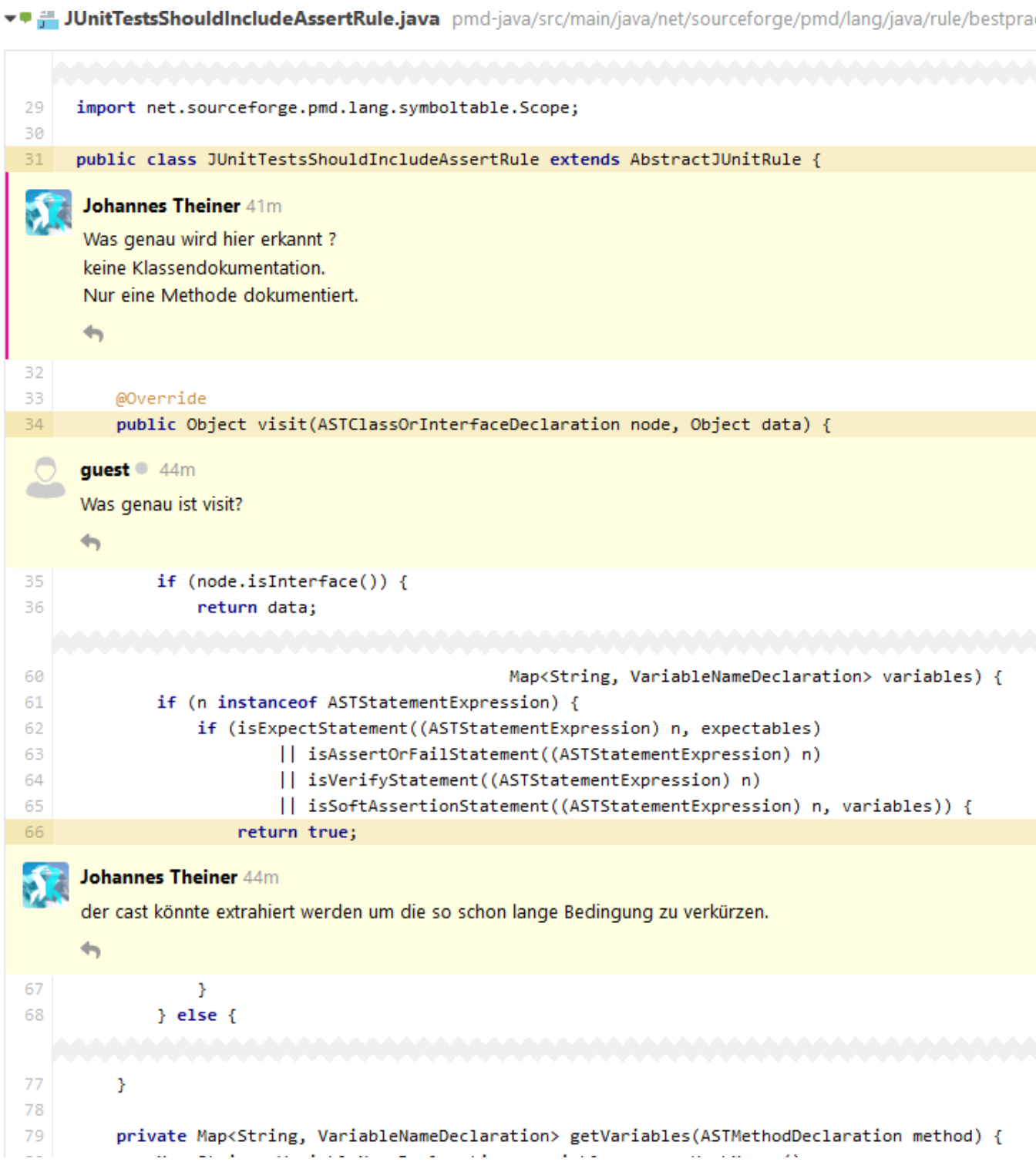
Nachteile:

- Die Planung beschränkt sich auf die Auswahl der Gutachter und der Festlegung des Abgabetermins.
- Die Ergebnisse werde oft nicht zwischen den Gutachtern ausgetauscht. Somit wird nicht sichergestellt, dass vermitete Fehler und Defekte identifiziert werden.

Der große Vorteil bei dieser Reviewart sind die geringen Kosten, dies ist für ein Open-Source Projekt gut geeignet. Aber das sicherstellen von Fehlern und Defekten ist oft gering, da die Ergebnisse zwischen den Gutachtern nicht ausgetauscht werden. Für PMD nicht geeignet.

# 3 Code-Reviews Durchführung

Gewählt wurde die Inspektion, durchgeführt wurde diese mit dem Tool Upsource[1].




[1] https://upsource.joethei.xyz/pmd/revision/4d2749b6974da31bf50902bb636162f6db02ec44?commentId=c088ccf5-5880-4ad9-a9f2-71a1ae555c2d

```
80          Map<String, VariableNameDeclaration> variables = new HashMap<>();
81          for (VariableNameDeclaration vnd : method.getScope().getDeclarations(VariableNameDeclar
82              variables.put(vnd.getName(), vnd);
83          }
84          return variables;
85      }
```

**Johannes Theiner** 23m

Warum ist das hier implementiert und nicht an einer generelleren Stelle ?

```
86
87      /**
99          for (Map.Entry<NameDeclaration, List<NameOccurrence>> entry : decls.entrySet()) {
100             Node parent = entry.getKey().getNode().getParent().getParent().getParent();
```

**Johannes Theiner** 1h

was ist parent genau ?, bzw welcher parent ?

1 2

```
101             if (parent.hasDescendantOfType(ASTMarkerAnnotation.class)
102                     && parent.getFirstChildOfType(ASTFieldDeclaration.class) != null) {
108                 Node type = parent.getFirstDescendantOfType(ASTReferenceType.class);
109                 if (!"ExpectedException".equals(type.getChild(0).getImage())) {
```

**guest** 28m

getChild(0) kann eine index out of bounce exception ergeben

```
110                     continue;
111                 }
135     }
136
137     /**
138      * Tells if the expression is an assert statement or not.
139      */
```

**Johannes Theiner** 39m

Kommentar nicht ganz korrekt.
Methodenname sagt an sich das selbe aus.

```
140     private boolean isAssertOrFailStatement(ASTStatementExpression expression) {
```

**guest** 38m
Abfragen können zusammengefasst werden

```
141         if (expression != null) {
142             ASTPrimaryExpression pe = expression.getFirstChildOfType(ASTPrimaryExpression.class
```

```
158      * Tells if the expression is verify statement or not
159      */
160     private boolean isVerifyStatement(ASTStatementExpression expression) {
```

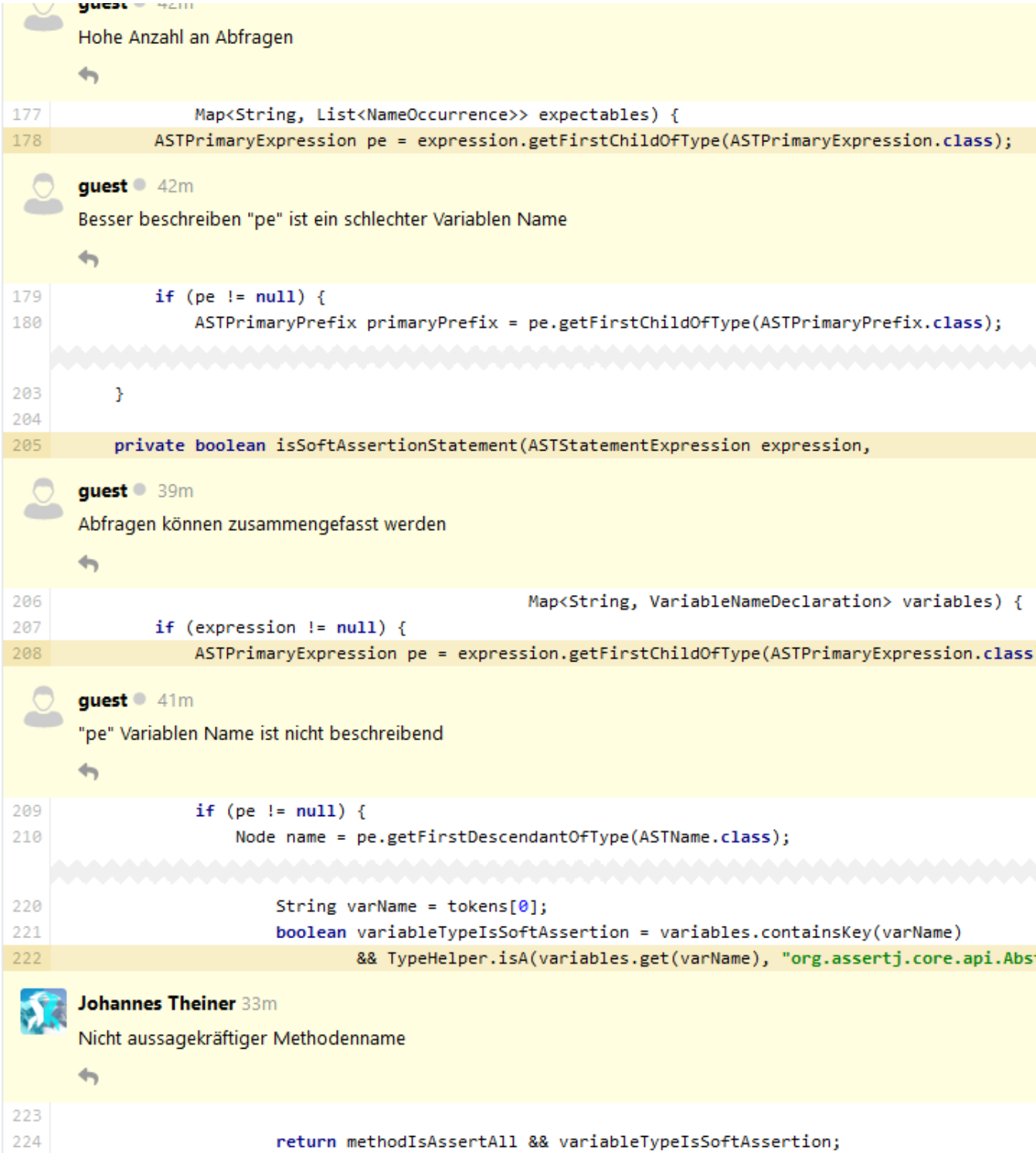
**guest** 39m
Abfragen können zusammengefasst werden

```
161         if (expression != null) {
162             ASTPrimaryExpression pe = expression.getFirstChildOfType(ASTPrimaryExpression.class
163             if (pe != null) {
164                 Node name = pe.getFirstDescendantOfType(ASTName.class);
165                 if (name != null) {
166                     String img = name.getImage();
167                     if (img != null && (img.startsWith("verify") || img.startsWith("Mockito.ver
168                         return true;
169                     }
170                 }
```

**Johannes Theiner** 42m
nur diese Zeilen unterschiedlich zur vorherigen Methode isAssertOrFailStatement, das kann genereller gefast werd

```
171             }
172         }
173         return false;
174     }
175
176     private boolean isExpectStatement(ASTStatementExpression expression,
```

**guest** 42m


Hohe Anzahl an Abfragen
Map<String, List<NameOccurrence>> expectables) {
ASTPrimaryExpression pe = expression.getFirstChildOfType(ASTPrimaryExpression.class);
guest 42m
Besser beschreiben "pe" ist ein schlechter Variablen Name
if (pe != null) {
ASTPrimaryPrefix primaryPrefix = pe.getFirstChildOfType(ASTPrimaryPrefix.class);
}
private boolean isSoftAssertionStatement(ASTStatementExpression expression,
guest 39m
Abfragen können zusammengefasst werden
Map<String, VariableNameDeclaration> variables) {
if (expression != null) {
ASTPrimaryExpression pe = expression.getFirstChildOfType(ASTPrimaryExpression.class
guest 41m
"pe" Variablen Name ist nicht beschreibend
if (pe != null) {
Node name = pe.getFirstDescendantOfType(ASTName.class);
String varName = tokens[0];
boolean variableTypeIsSoftAssertion = variables.containsKey(varName)
&& TypeHelper.isA(variables.get(varName), "org.assertj.core.api.Abs
Johannes Theiner 33m
Nicht aussagekräftiger Methodenname
return methodIsAssertAll && variableTypeIsSoftAssertion;