MGL7460 — Automne 2016

Outils d'analyse du code

Guy Tremblay
Professeur

Département d'informatique
UQAM
http://www.labunix.uqam.ca/~tremblay

10 novembre 2016

Contenu

1. Introduction

Il existe de nombreux outils d'analyse de code

Analyse de code = analyse de la qualité du code

Il existe de nombreux outils d'analyse de code

Analyse de code = analyse de la qualité du code

Certains outils sont spécifiques à Java, mais plusieurs peuvent analyser du code dans des langages variés

Ces outils sont généralement complémentaires

Conventions

Potential bugs

Bad practices

2. CheckStyle


checkstyle


checkstyle

Qu’est-ce que Checkstyle ?

Checkstyle est un outil de contrôle de code, utilisé en développement de logiciel. Il permet de vérifier le style d’un code source écrit en langage Java.

Source: `https://fr.wikipedia.org/wiki/Checkstyle`

Checkstyle définit des règles

Checkstyle définit un ensemble de modules contenant des règles que l'on peut configurer de façon plus ou moins stricte.

Checkstyle définit des règles

Checkstyle définit un ensemble de modules contenant des règles que l'on peut configurer de façon plus ou moins stricte.

Chaque règle peut se traduire, selon le cas, par une notification, un avertissement ou par une **erreur**.

Source: `https://fr.wikipedia.org/wiki/Checkstyle`

Exemples de vérifications effectuées par Checkstyle

- présence de commentaires Javadoc
- conventions de nommage des attributs et des méthodes
- nombre de paramètres de méthodes
- longueur des lignes
- présence d'en-têtes obligatoires
- utilisation des importations de paquets, de classes, des modificateurs de portée et des blocs d'instructions
- espacement entre certains caractères
- bonnes pratiques d'écriture de classe
- sections de code dupliqué
- diverses mesures de complexité

Source: `https://fr.wikipedia.org/wiki/Checkstyle`

Exemples de vérifications effectuées par Checkstyle

- présence de commentaires Javadoc
- conventions de nommage des attributs et des méthodes
- nombre de paramètres de méthodes
- longueur des lignes
- présence d'en-têtes obligatoires
- utilisation des importations de paquets, de classes, des modificateurs de portée et des blocs d'instructions
- espacement entre certains caractères
- bonnes pratiques d'écriture de classe
- sections de code dupliqué
- diverses mesures de complexité

Complexité cyclomatique : *Generally 1–4 is considered good, 5–7 ok, 8–10 consider re-factoring, and 11+ re-factor now !*

Source: `https://fr.wikipedia.org/wiki/Checkstyle`

3. PMD


pmd
DON'T SHOOT THE MESSENGER

Version initiale : Java seulement

PMD est un framework qui permet d'analyser le code source Java. Il contient un certain nombre de règles qui assurent la qualité de code : le code inutile, les imbrications trop complexes...

PMD apporte une série d'outils complémentaires :

- un détecteur de code dupliqué par copier-coller [. . .],
- un analyseur de flux de données,
- un détecteur de code mort

Source: `https://fr.wikipedia.org/wiki/PMD_(logiciel)`

Analyse de flux de données

```
public class Foo {
  public void foo() {
    int buz = 5;
    buz = 6;
    foo(buz);
    buz = 2;
  }
}
```

Source: http://pmd.github.io/pmd-4.2.5/rules/controversial.html

Analyse de flux de données

```
public class Foo {
  public void foo() {
    int buz = 5;  // Pourquoi initialiser...
    buz = 6;      // puis redéfinir?
    foo(buz);
    buz = 2;      // Pourquoi affecter?
  }
}
```

Source: http://pmd.github.io/pmd-4.2.5/rules/controversial.html

Analyse de code mort (*dead code*)

Protected or private methods that are never used by any classes in the same project are strongly suspected to be dead code.

Dead code means unnecessary, inoperative code that should be removed.

This helps in maintenance by decreasing the maintained code size, making it easier to understand the program.

Source: *http://www.sonarqube.org/*

detect-dead-code-and-calls-to-deprecated-methods-with-sonar-squid

Analyse de code mort (*dead code*)

Protected or private methods that are never used by any classes in the same project are strongly suspected to be dead code.

Dead code means unnecessary, inoperative code that should be removed.

This helps in maintenance by decreasing the maintained code size, making it easier to understand the program.

Source: `http://www.sonarqube.org/detect-dead-code-and-calls-to-deprecated-methods-with-sonar-squid`

PMD peut aussi traiter divers langages

Versions plus récentes

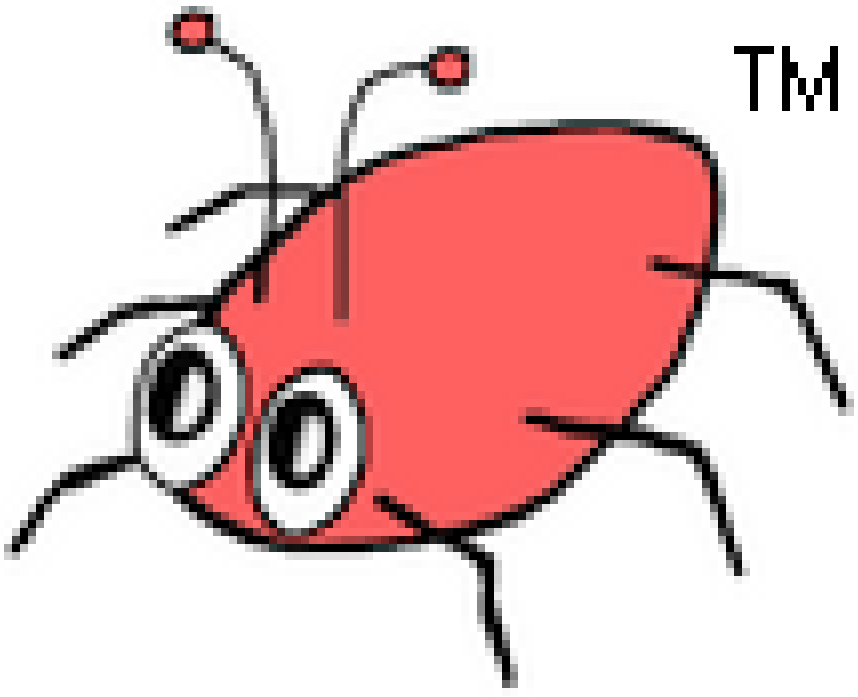
- Java
- JavaScript
- Salesforce.com Apex
- PLSQL
- Apache Velocity
- XML
- XSL

Et PMD peut analyser le code dupliqué dans de nombreux langages

Utilise CPD = *Copy Paste Detector*

- Java
- C
- C++
- C#
- PHP
- Ruby
- Fortran
- JavaScript
- PLSQL
- Apache Velocity
- Ruby
- Scala
- Objective C
- Matlab
- Python
- Go
- Swift
- Salesforce.com Apex

4. FindBugs



TM

Qu’est-ce que FindBugs ?

FindBugs est un logiciel libre d’analyse statique **de bytecode Java**. Son but est de trouver des bugs dans les programmes Java en identifiant des patterns reconnus comme étant des bugs.

Source: `https://fr.wikipedia.org/wiki/FindBugs`

Qu’est-ce que FindBugs ?

FindBugs est un logiciel libre d’analyse statique **de bytecode Java**. Son but est de trouver des bugs dans les programmes Java en identifiant des patterns reconnus comme étant des bugs.

Source: `https://fr.wikipedia.org/wiki/FindBugs`

Les bogues sont classés selon différentes catégories

- *Bad practice*
- *Correctness*
- *Dodgy code*
- *Experimental*
- *Internationalization*
- *Malicious code vulnerability*
- *Multithreaded correctness*
- *Performance*
- *Security*

Le nombre de bogues décrits est. . .

Version 3.0.1 : `http://findbugs.sourceforge.net/bugDescriptions.html`

FindBugs Bug Descriptions

This document lists the standard bug patterns reported by FindBugs version 3.0.1.

Summary

Description	Category
BC: Equals method should not assume anything about the type of its argument	Bad practice
BIT: Check for sign of bitwise operation	Bad practice
CN: Class implements Cloneable but does not define or use clone method	Bad practice
CN: clone method does not call super.clone()	Bad practice
CN: Class defines clone() but doesn't implement Cloneable	Bad practice
CNT: Rough value of known constant found	Bad practice
Co: Abstract class defines covariant compareTo() method	Bad practice
Co: compareTo()/compare() incorrectly handles float or double value	Bad practice
Co: compareTo()/compare() returns Integer.MIN_VALUE	Bad practice
Co: Covariant compareTo() method defined	Bad practice
DE: Method might drop exception	Bad practice
DE: Method might ignore exception	Bad practice
DMI: Adding elements of an entry set may fail due to reuse of Entry objects	Bad practice
DMI: Random object created and used only once	Bad practice
DMI: Don't use removeAll to clear a collection	Bad practice
Dm: Method invokes System.exit(...)	Bad practice
Dm: Method invokes dangerous method runFinalizersOnExit	Bad practice
ES: Comparison of String parameter using == or !=	Bad practice
ES: Comparison of String objects using == or !=	Bad practice
Eq: Abstract class defines covariant equals() method	Bad practice
Eq: Equals checks for incompatible operand	Bad practice
Eq: Class defines compareTo(...) and uses Object.equals()	Bad practice
Eq: equals method fails for subtypes	Bad practice
Eq: Covariant equals() method defined	Bad practice
FI: Empty finalizer should be deleted	Bad practice
FI: Explicit invocation of finalizer	Bad practice
FI: Finalizer nulls fields	Bad practice
FI: Finalizer only nulls fields	Bad practice
FI: Finalizer does not call superclass finalizer	Bad practice
FI: Finalizer nullifies superclass finalizer	Bad practice
FI: Finalizer does nothing but call superclass finalizer	Bad practice
FS: Format string should use %n rather than \n	Bad practice
GC: Unchecked type in generic call	Bad practice
HE: Class defines equals() but not hashCode()	Bad practice

Le nombre de bogues décrits est. . . très grand

Version 3.0.1 : `http://findbugs.sourceforge.net/bugDescriptions.html`

424

Un exemple de bug — non trivial — pouvant être identifié

```
int inter1(boolean b) {
  Object x = null;
  if (b) x = new Object();
  return helper1(x, b);
}


private int helper1(Object x, boolean b) {
  if (b) return 0;
  return x.hashCode();
}
```

Source: «*Finding More Null Pointer Bugs, But Not Too Many*», Hovemeyer & Pugh

Un exemple de bug — non trivial — pouvant être identifié

```java
// Bug when b is false
int inter1(boolean b) {
  Object x = null;
  if (b) x = new Object();
  return helper1(x, b);
}

// Bug when x is null and b is false
private int helper1(Object x, boolean b) {
  if (b) return 0;
  return x.hashCode();
}
```

Source: «*Finding More Null Pointer Bugs, But Not Too Many*», Hovemeyer & Pugh

5. SonarQube

SonarQube

http://www.sonarqube.org/

Qu’est-ce que SonarQube ?

SonarQube (précédemment Sonar) est un logiciel libre permettant de mesurer la qualité du code source en continu.

Source: `https://fr.wikipedia.org/wiki/SonarQube`

SonarQube met l'accent sur la «dette technique»

At the heart of SonarQube is the concept of technical debt : the cumulative cost of work that's been put off or done poorly enough that it needs to be refactored.

Because measuring technical debt is a core principle of SonarQube, it's not surprising that its creators would come up with a set of technical debt metrics—in dollars and days.

Source: *«SonarQube in Action», Campbell & Papapetrou*

SonarQube supporte de nombreux langages

Peut traiter plus de vingt-cinq langages ([ABAP], Java, C/C++, Objective-C, C#, PHP, Flex, Groovy, JavaScript, [PL/I], Python, PL/SQL, COBOL, [Visual Basic], . . .), dont certains sont sous licence commerciale.

Source: `https://fr.wikipedia.org/wiki/SonarQube`

SonarQube supporte de nombreux langages

Peut traiter plus de vingt-cinq langages ([ABAP], Java, C/C++, Objective-C, C#, PHP, Flex, Groovy, JavaScript, [PL/I], Python, PL/SQL, COBOL, [Visual Basic], ...), dont certains sont sous licence commerciale.

Source: `https://fr.wikipedia.org/wiki/SonarQube`

Note :

- Pas de support pour Ruby ☹
  Voir plutôt `rubocop` :
  `https://github.com/bbatsov/rubocop`

SonarQube effectue de nombreuses analyses… puisqu'il intègre de nombreux autres outils

Notamment : CheckStyle, FindBugs, PMD, etc.

- identification des duplications de code
- mesure du niveau de documentation
- respect des règles de programmation
- détection des bugs potentiels
- évaluation de la couverture de code par les tests unitaires
- analyse de la répartition de la complexité
- analyse du design et de l'architecture d'une application

Source: `https://fr.wikipedia.org/wiki/SonarQube`

SonarQube effectue des analyses selon «Sept axes de qualité»

Seven Axes of Quality = Seven Deadly Developer Sins

- *Potential bugs*
- *Coding rules*
- *Tests*
- *Duplication*
- *Comments*
- *Architecture and design*
- *Complexity*

Source: *«SonarQube in Action», Campbell & Papapetrou*

Potential bugs and coding rules

The creators of SonarQube list potential bugs and coding rules as separate axes, but for reporting they group them together under **issues**. *Generally, you can consider issue counts as lagging quality indicators ; they show what's already gone wrong.*

Potential bugs and coding rules

The creators of SonarQube list potential bugs and coding rules as separate axes, but for reporting they group them together under **issues**. *Generally, you can consider issue counts as lagging quality indicators ; they show what's already gone wrong.*

It's clear that some issues are worse than others. That's why SonarQube ranks them at different severities : Blocker, Critical, Major, Minor, and Info.

Source: *«SonarQube in Action», Campbell & Papapetrou*

Either remove or fill this block of code. ...

Code Smell Major Open Not assigned 5min effort

NullPointerException might be thrown as 's' is nullable here ...

Bug Blocker Open Not assigned 10min effort

Document this public method. ...

Code Smell Minor Open Not assigned 10min effort

NullPointerException might be thrown as 's' is nullable here ...

Bug Blocker Open Not assigned 10min effort

Document this public method. ...

Code Smell Minor Open Not assigned 10min effort

Either remove or fill this block of code. ...

Code Smell Major Open Not assigned 5min effort

Move the "" string literal on the left side of this string comparison. ...

Code Smell Major Open Not assigned 10min effort

SonarQube effectue des analyses selon «Sept axes de qualité»

Seven Axes of Quality = Seven Deadly Developer Sins

- *Potential bugs*
- *Coding rules*
- *Tests*
  Pourcentage des tests unitaires exécutés avec succès
  Pourcentage de couverture des tests
- *Comments*
- *Duplication*
- *Architecture and design*
- *Complexity*

Source: *«SonarQube in Action», Campbell & Papapetrou*

Line Coverage	36.4%
Condition Coverage	6.1%
Uncovered Lines	154
Uncovered Lines on New Code	0
Uncovered Conditions	77
Uncovered Conditions on New Code	0
Lines to Cover on New Code	0
Uncovered Lines by IT on New Code	0
Uncovered Conditions by IT on New Code	0
Lines to Cover by IT on New Code	0
Lines to Cover	242

```
13    public static int longueur2( String s ) {
14      if ( s == null )
15        return s.length();
16
17      return 0;
18    }
19
20    public static int longueur3( String s ) {
21      if ( s == null )
22        return 0;
23
24      if ( s.equals( "" ) ) {
25      }
26
27      return s.length();
28    }
29
30    public static int longueur4( String s ) {
31      int n = 0;
32      for( char c : s.toCharArray() ) {
33        n += 1;
34      }
35
```

SonarQube effectue des analyses selon «Sept axes de qualité»

Seven Axes of Quality = Seven Deadly Developer Sins

- *Potential bugs*
- *Coding rules*
- *Tests*
- *Comments*
  Pourcentage d'items documentés
- *Duplication*
- *Architecture and design*
- *Complexity*

Source: *«SonarQube in Action», Campbell & Papapetrou*

SonarQube effectue des analyses selon «Sept axes de qualité»

Seven Axes of Quality = Seven Deadly Developer Sins

- *Potential bugs*
- *Coding rules*
- *Tests*
- *Comments*
- *Duplication*
  Nombre de lignes, blocs ou fichiers dupliqués (copiés/collés ?)
- *Architecture and design*
- *Complexity*

Source: *«SonarQube in Action», Campbell & Papapetrou*

SonarQube effectue des analyses selon «Sept axes de qualité»

Seven Axes of Quality = Seven Deadly Developer Sins

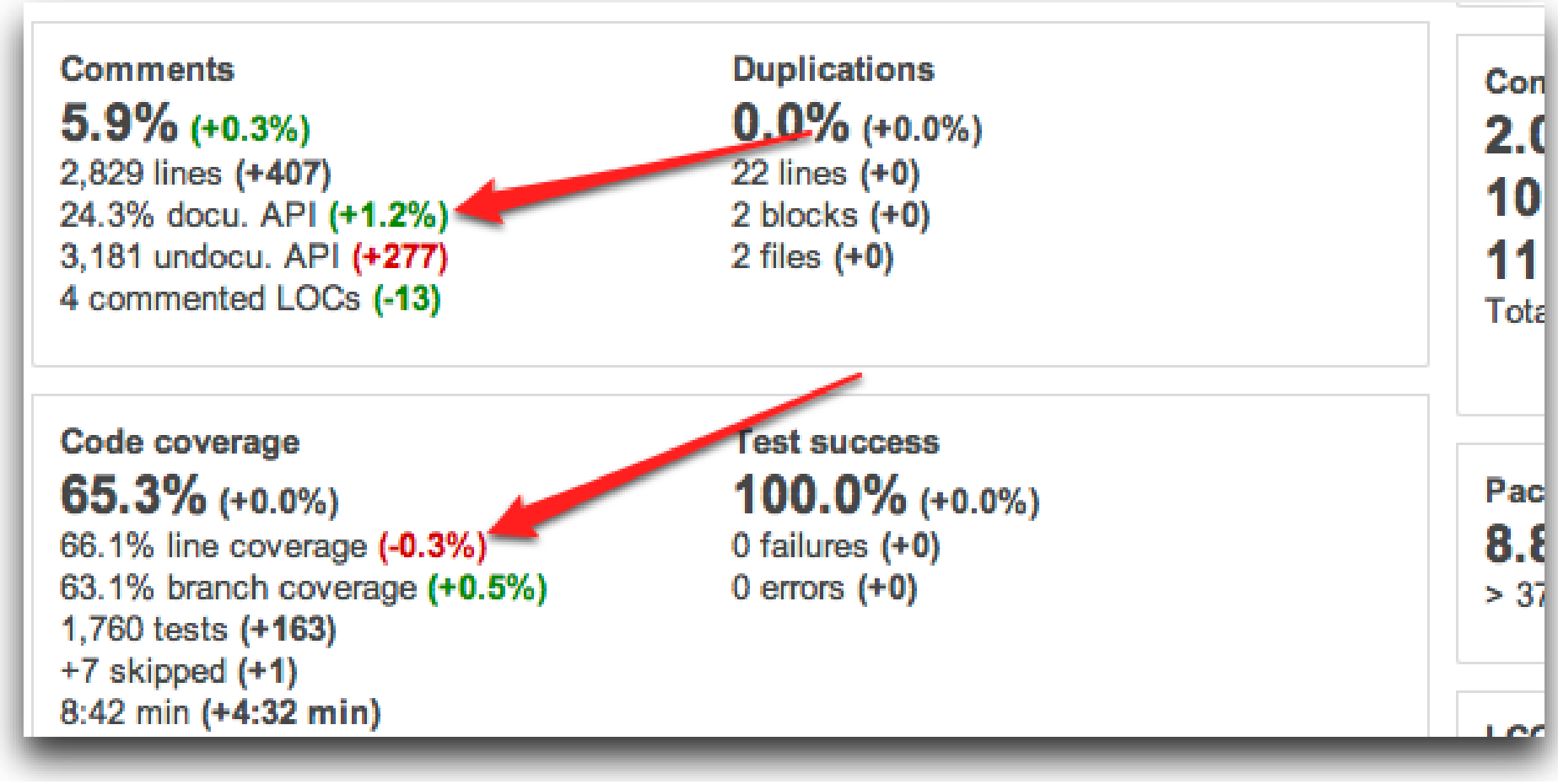
- *Potential bugs*
- *Coding rules*
- *Tests*
- *Comments*
- *Duplication*
- *Architecture and design*
- *Complexity*
  ≈ *Complexité cyclomatique*

Source: *«SonarQube in Action», Campbell & Papapetrou*

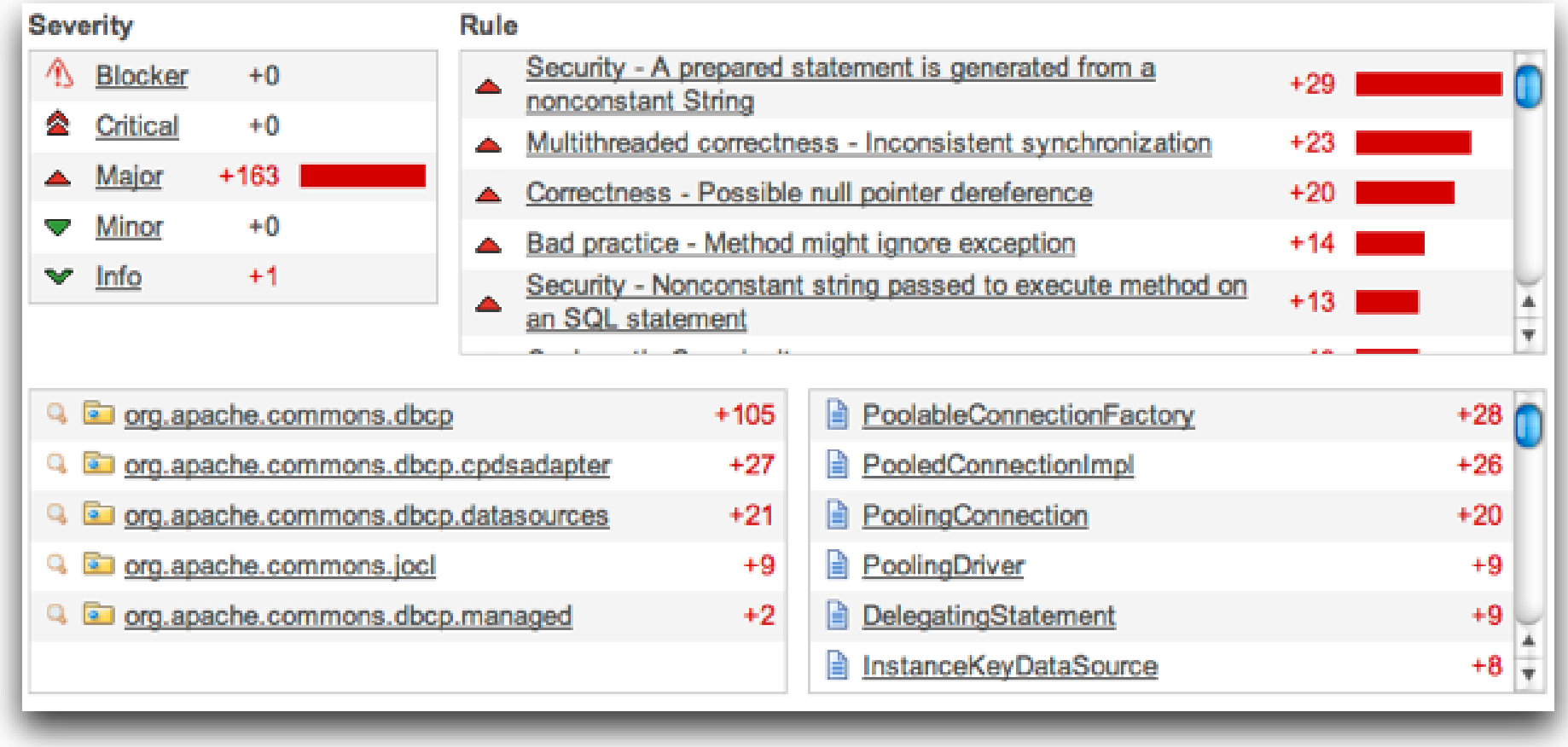
SonarQube possède de nombreuses autres fonctionnalités

- Evolution dans le temps et vues différentielles
- Analyses entièrement automatisées : integration avec Maven, Ant, Gradle et serveurs d'intégration continue (Atlassian Bamboo, Jenkins, Hudson, . . .).
- Intégration avec l'environnement de développement Eclipse
- Intégration avec des outils externes : JIRA, Mantis, LDAP, Fortify. . .
- Extensible par des plugins
- Implémentation de SQALE pour évaluer la dette technique

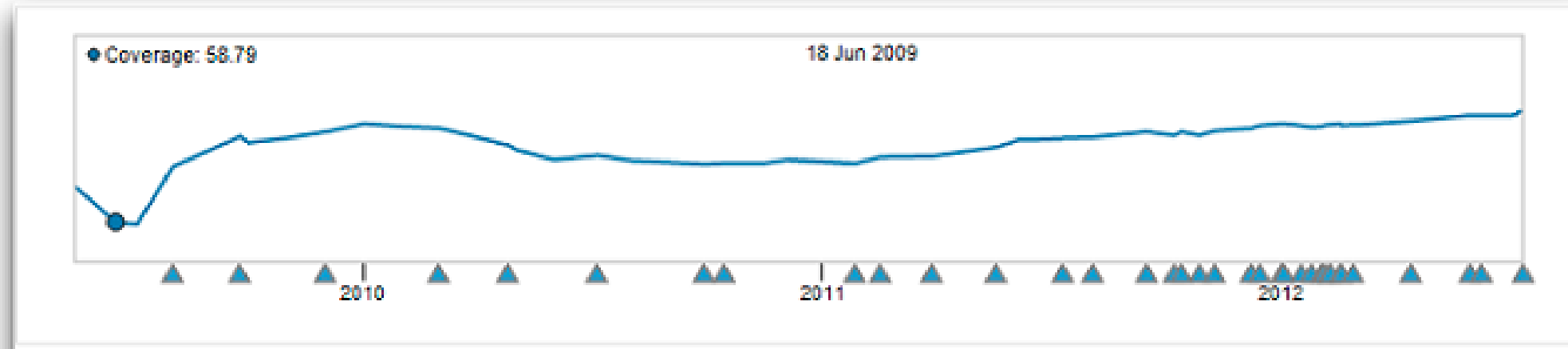
Source: `https://fr.wikipedia.org/wiki/SonarQube`



Source: `http://www.sonarqube.org/differentials-four-ways-to-see-whats-changed/`



Source: `http://www.sonarqube.org/differentials-four-ways-to-see-whats-changed/`



	17 May 2009 1.9	11 Jul 2011 2.9	04 Aug 2011 2.10	15 Sep 2011 2.11	08 Nov 2011 2.12	02 Jan 2012 2.13	26 Feb 2012 2.14	12 Apr 2012 3.0	29 May 2012 3.1-SNAPSHOT	10 Jul 2012 3.2-SNAPSHOT
Coverage	62.8%	68.1%	68.2%	68.8%	68.9%	69.7%	69.6%	70.0%	70.6%	71.1%

Source: `http://www.sonarqube.org/differentials-four-ways-to-see-whats-changed/`

SonarQube possède aussi de nombreuses autres fonctionnalités

- Evolution dans le temps et vues différentielles
- Analyses entièrement automatisées : integration avec Maven, Ant, Gradle et serveurs d'intégration continue (Atlassian Bamboo, Jenkins, Hudson, . . .).
- Intégration avec l'environnement de développement Eclipse
- Intégration avec des outils externes : JIRA, Mantis, LDAP, Fortify. . .
- Extensible par des plugins
- Implémentation de SQALE pour évaluer la dette technique

Source: `https://fr.wikipedia.org/wiki/SonarQube`

Références

J.A. Campbell and P.P. Papatreou.
SonarQube In Action.
Manning, 2013.