

## Chapitre 2

# Bonnes pratiques pour améliorer la qualité du code Java



Mohamed Romdhani  
Avril 2017

## Sommaire

Amélioration de la qualité du code

- Règles du "Clean Code" en Java
- Contrôle du style de codage Java avec Checkstyle
- Tests unitaire avec JUnit et Couvertures de tests avec Corbertura
- Analyse statique du code avec PMD et FindBugs
- Métriques du code
- Indicateurs de qualité avec SonarQube

## Règles du "Clean Code" en Java

### Pourquoi un Clean Code ?

Amélioration de la qualité du code

- Pour s'adapter au changement à moindre coût
- Pour pérenniser les produits
- Pour valoriser le travail d'un développeur
- Vecteur de motivation intrinsèque
- Pour réconcilier progrès économique et social

*[Kent Beck, Extreme Programming Explained 2nd Ed]*

## Qu'est-ce qu'un Code Clean ?

- **Testé !**
- **Elégant et facile à lire**
- **Montre clairement l'intention de son auteur**
  - Ecrit à destination des prochains développeurs
  - Il est difficile pour les bugs de se cacher
- **Simple**
  - Non dupliqué
  - Contient un minimum d'artefacts (classes, méthodes, etc.)
- **Les dépendances sont minimales et explicites**  
*[Bjarne Stroustrup, Grady Booch, Ron Jeffries, Kent Beck, Michael Feathers, Ward Cunningham...]*

## Quelles horreurs dans le code ?

### Code Smells

Large class  
Data class  
**Duplicated code**  
Uncommunicative name  
Lazy class  
Refused bequest  
Type embedded in name  
Message chain  
Conditional complexity  
Inappropriate intimacy  
Data clumps  
**Dead code**  
Speculative generality  
Comments  
Primitive obsession  
Shotgun Surgery  
Middle man  
Inconsistent names  
Divergent change  
Feature envy  
Long method  
Temporary field  
Long parameter list  
Alternative classes with different interfaces  
**Wrong level of abstraction**  
Parallel inheritance hierarchies

## Quelles horreurs dans le code ?

- **S**ingleton
- **T**ight coupling
- **U**ntestable
- **P**remature optimization
- **I**ndescriptive naming
- **D**uplication

## Quelles bonnes pratiques ?

- **SOLID**
- **DRY**
- **YAGNI**
- **KISS**
- **GRASP**
  - **General Responsibility Assignment Software Patterns** (or principles), consiste en un semble de guides pour assigner la responsabilité aux classes et objets en Conception OO
- **SoC : Separation of Concerns**
- **Loi de Demeter**
  - Chaque unité doit avoir une connaissance limitée des autres unités : elle doit voir que les unités étroitement liées à l'unité actuelle

## Rules

Amélioration de la qualité du code

- **The Boy Scout rule :**
  - Leave the campground cleaner than you found it.
- **The Clean Code rule :**
  - Leave the code cleaner than you found it.
- **By following this rule, code gets better over time.**

M.Romdhani, Avril 2017

9

## Meaningful Names

Amélioration de la qualité du code

- **Les noms doivent révéler l'intention**
- **Ne tronquez pas les mots !**
  - Le coût du
- **Utilisez des mots qui ont du sens**
- **Pas d'encodage**
- **Ne surchargez pas les noms d'infos inutiles**
- **N'hésitez pas à changer un nom**

M.Romdhani, Avril 2017

10

## Meaningful Names

- **Use intention revealing names**

```
int d // wtf ?  
int m_iHwWindowHandle // wtf ?  
int loopCounter  
int windowHandle
```

- **Avoid Disinformation**

No s\*\*t Sherlock. I want to code, not solve puzzles  
Avoid inconsistent spelling.

- **Use Pronounceable Names**

```
txtPwordCfrm ( go on say it !!)  
passwordConfirmTextBox
```

## Meaningful Names

- **Avoid mental mapping don't make me think**

```
not-null="true"  
not-null="false"  
not-null="wtf?"
```

```
null="true" allow-nulls="false" //easy ...
```

- **Avoid single letter variable names**

- **Use precious brain time to solve the problem, not solve puzzles in the code**

- **Use nouns in class names, don't use verbs**

## Meaningful Names

- **Method names should have verbs**
  - should, get , is, has
- **Don't be cute**
  - whack(), kill(), finger(), mount(), hopeORACSave(), diediedie()
- **Pick one word per concept**
  - Don't use the same word for different things
  - Be consistent. Add() always adds.
- **Use Problem Domain Names**

## Functions

- **Small**
- **How small ?**
- **Smaller than that**
- **What's the smallest function you can think of ?**
- **That's how small.**

## Functions

- Avoid more than one if/else in a function
- Should not be large enough to hold nested structures.
- A function should not many indentation levels.
- Do one thing. Do it well . Should do it only

## Functions

- One Level of Abstraction per Function

```
buildPage(){
    buildHeader();
    buildBody();
}
buildHeader(){
    buildMetaTags()
    buildTitle()
}
buildMetaTag(){
    writer.Render("<meta />")
}
buildTitle(){
    writer.Render("<title />")
}
```



## Function Arguments

- Ideally zero arguments
- One, two are ok
- Avoid three or more arguments
- Have no side effects
  - Promise to do one thing but don't do other things
- Command Query Separation
  - Do something or answer something not both

## The Law of Demeter

- A method *f* of class *C* should only call
  - *C*
  - an object created by *f*
  - An object passed as an argument to *f*
  - An object held in an instance variable of *C*

## Comments

Amélioration de la qualité du code

- Are bad
- Explain yourself in code, not comments
- If you have to comment it, extract into a function and give a descriptive name
- Code that you have to comment should be a function
- Don't mumble.
- No TODOs or //change before going to production
- Don't comment code, delete it. Use version control

M.Romdhani, Avril 2017

19

## Classes

Amélioration de la qualité du code

- First rule of classes is that they should be small
- Second rule of classes is that they should be smaller than that
- How small ?
- Count responsibilities
- How many things is it doing ?

M.Romdhani, Avril 2017

20

## Single Responsibility Principle

- A class should have only one reason to *change*
- What will make the class change ?
- Note what can cause the Sudoku class to change .

## A design is simple

- If it runs all tests
- If it contains no duplication
- If it expresses the intent of the programmer
- If it minimizes the number of classes and methods

Look back at your code. Is it simple ?

## Successive refinement

- By applying simple refactoring, cleaner code emerges over time.
- You can't fix it all in big step.
- Be patient, take small steps.

## Contrôle du style de codage Java avec 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.**
  - Les vérifications de Checkstyle portent essentiellement sur la forme et ne permettent en rien de dire qu'un programme est correct ou complet.
    - En pratique, il est très fastidieux de respecter l'ensemble de toutes les contraintes de style que l'on peut imposer au travers de checkstyle. Ces contraintes peuvent par ailleurs nuire à la dynamique des étapes de programmation. Il s'agit donc de déterminer, selon le type du développement et la qualité que l'on attend, quel doit être le niveau de vérification.
- **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.



## Modules de Checkstyle

- **Checkstyle permet, par exemple, de vérifier :**
  - la présence de commentaires Javadoc pour les classes, les attributs et les méthodes
  - les conventions de nommage des attributs et des méthodes
  - une limitation du nombre de paramètres de méthodes, la longueur des lignes, etc.
  - la présence d'en-têtes obligatoires
  - l'utilisation des importations de paquets, de classes, des modificateurs de portée et des blocs d'instructions
  - l'espacement entre certains caractères
  - les bonnes pratiques d'écriture de classe
  - les sections de code dupliqué
  - diverses mesures de complexité, notamment des expressions

## Utilisation de Checkstyle

Amélioration de la qualité du code

- Checkstyle se présente sous la forme d'une archive Jar qui peut être exécutée directement sur une machine virtuelle Java ou utilisé dans une tâche ANT. Checkstyle peut également être intégré à des environnements de développement intégrés et d'autres outils comme NetBeans, Eclipse, jEdit, etc.
- Un plugin Checkstyle permet, par exemple,
  - de surcharger la coloration syntaxique ou les décorations dans l'éditeur de code
  - de décorer l'explorateur de projets au niveau des ressources posant problème
  - de rajouter des entrées dans les vues d'erreurs et d'avertissement.
- Le développeur peut ainsi accéder directement aux parties de code qui ne respectent pas le style choisi.

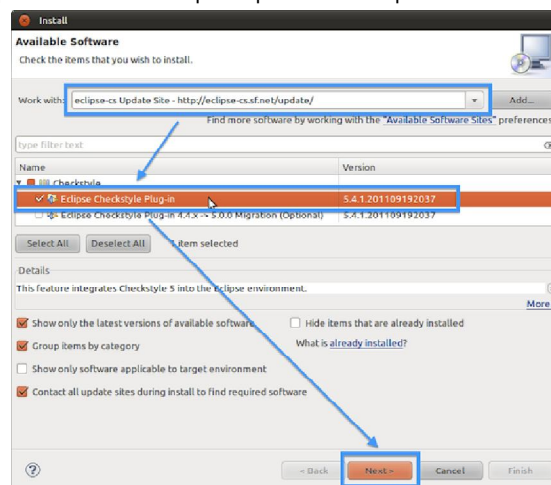
M.Romdhani, Avril 2017

27

## Utilisation de CheckStyle sous Eclipse

Amélioration de la qualité du code

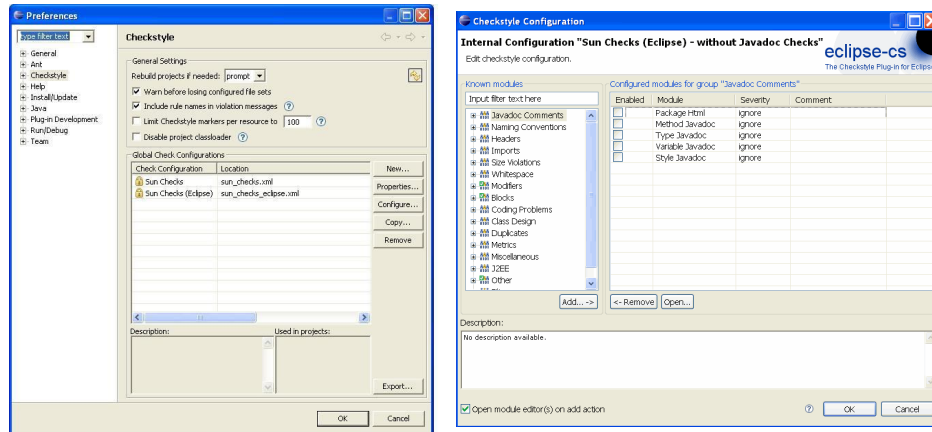
- Installation du plugin Eclipse-cs :
  - Update URL : Enter <http://eclipse-cs.sf.net/update/>



M.Romdhani, Avril 2017

28

## Configuration des règles sous Eclipse



M.Romdhani, Avril 2017

29

## Analyse statique du code avec PMD et FindBugs

## PMD static analysis tool

Amélioration de la qualité du code

- **PMD is a static analysis tool for the Java programming language**
- **PMD means**
  - Programming Mistake Detector
- **PMD scans Java source code and looks for potential problems like:**
  - Possible bugs - empty try/catch/finally/switch statements
  - Dead code - unused local variables, parameters and private methods
  - Suboptimal code - wasteful String/StringBuffer usage
  - Overcomplicated expressions - unnecessary if statements, for loops that could be while loops
  - Duplicate code - copied/pasted code means copied/pasted bugs
- **PMD can be run**
  - Separately
  - As a plugin in NetBeans and other IDEs
- **References**
  - <http://pmd.sourceforge.net/>
  - [http://en.wikipedia.org/wiki/PMD\\_\(software\)](http://en.wikipedia.org/wiki/PMD_(software))



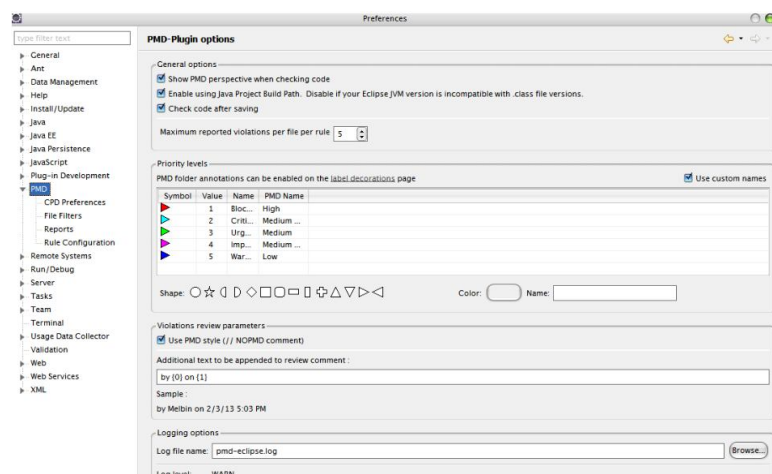
M.Romdhani, Avril 2017

31

## PMD Eclipse Configuration

Amélioration de la qualité du code

- **You can configure PMD from eclipse, Window->Preferences->Java->PMD**



M.Romdhani, Avril 2017

32



## Running PMD

Amélioration de la qualité du code

The screenshot shows the Eclipse IDE with a project named 'PMDTest'. The 'PMDTest.java' file is open, showing a class with three methods: 'main', 'CALL\_METHOD', and 'CallHello'. A context menu is open over the 'PMDTest' class, with the 'PMD' option highlighted. The 'PMD' option is part of a group of actions related to the PMD tool.

**Violations Overview**

Priority	Line	created	Rule	Error Message
9	9	Sun Feb 03 22:17:58 IST 2013	VariableNamingConventions	Variables should s
9	9	Sun Feb 03 22:17:58 IST 2013	MethodNamingConventions	Method names sh
9	9	Sun Feb 03 22:17:58 IST 2013	MethodNamingConventions	Method names sh
9	9	Sun Feb 03 22:17:58 IST 2013	VariableNamingConventions	Only variables tha
13	13	Sun Feb 03 22:17:58 IST 2013	MethodNamingConventions	Method names sh
14	14	Sun Feb 03 22:17:58 IST 2013	SystemPrintin	System.out.print it
13	13	Sun Feb 03 22:17:58 IST 2013	SystemPrintin	System.out.print it
13	13	Sun Feb 03 22:17:58 IST 2013	MethodArgumentCouldBeFinal	Parameter 'INPUT_
13	13	Sun Feb 03 22:17:58 IST 2013	MethodArgumentCouldBeFinal	Parameter 'args' is
13	13	Sun Feb 03 22:17:58 IST 2013	UseSingleton	All methods are st

M.Romdhani, Avril 2017 33

## What is Findbugs?

Amélioration de la qualité du code

- **Result of a research project at the University of Maryland**
  - Can detect many types of common, hard-to-find bugs
- **Static analysis tool for Java**
- **How to use FindBugs**
  - Standalone Swing application
  - Eclipse plug-in
  - Integrated into the build process (Ant or Maven)
- **How it works**
  - Use "bug patterns" to detect potential bugs

**NullPointerException**

```
Address address = client.getAddress();
if ((address != null) || (address.getPostCode() != null)) {
    ...
}
```

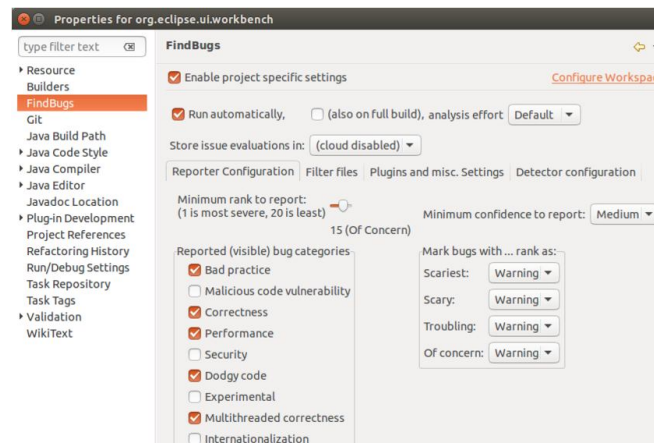
**Uninitialized field**

```
public class ShoppingCart {
    private List items;
    public addItem(Item item) {
        items.add(item);
    }
}
```

M.Romdhani, Avril 2017 34

## FindBugs Eclipse Configuration

- You can configure FindBugs from eclipse, Window->Preferences->Java->FindBugs



M.Romdhani, Avril 2017

35

## Bug Severity and Confidence

- **Bug severity** : As of version 2, FindBugs started ranking bugs with a scale from 1 to 20 to measure the severity of defects:
  - **Scariest**: ranked between 1 & 4.
  - **Scary**: ranked between 5 & 9.
  - **Troubling**: ranked between 10 & 14.
  - **Of concern**: ranked between 15 & 20.
- **Bug Confidence** : While the bug rank describes severity, the confidence factor reflects the likelihood of these bugs to be flagged as real ones. The confidence was originally called priority, but it was renamed in the new version

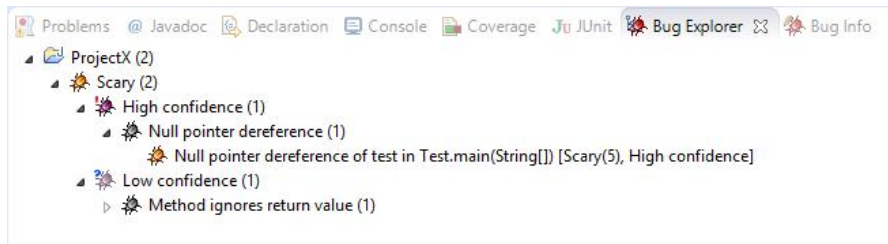
*Of course, some defects can be open to interpretation, and they can even exist without causing any harm to the desired behavior of a software. That's why, in a real world situation, we need to properly configure static analysis tools by choosing a limited set of defects to activate in a specific project.*

M.Romdhani, Avril 2017

36

## FindBugs Reports

- In order to launch a static analysis on a project using the FindBugs Eclipse plugin, you need to right-click the project in the package explorer, then, click on the option labeled find bugs.
- After launch, Eclipse shows the results under the Bug Explorer window as shown in the screenshot below:



## Tests unitaire avec JUnit et Couvertures de tests avec Corbertura

## Présentation de JUnit

Amélioration de la qualité du code

### ■ Framework de Tests Unitaires

- JUnit est écrit par Erich Gamma (of Design Patterns fame) et Kent Beck (creator of XP methodology)
- JUnit aide le programmeur à:
  - Définir et exécuter des tests et des suites de tests
  - Formaliser les exigences et clarifier l'architecture
  - Écrire et déboguer le code



### ■ Caractéristiques des tests unitaires

- Exécutable rapidement et automatiquement
- Indépendance
- Nombreux
- Écrit avant le code (recommandé ;-)

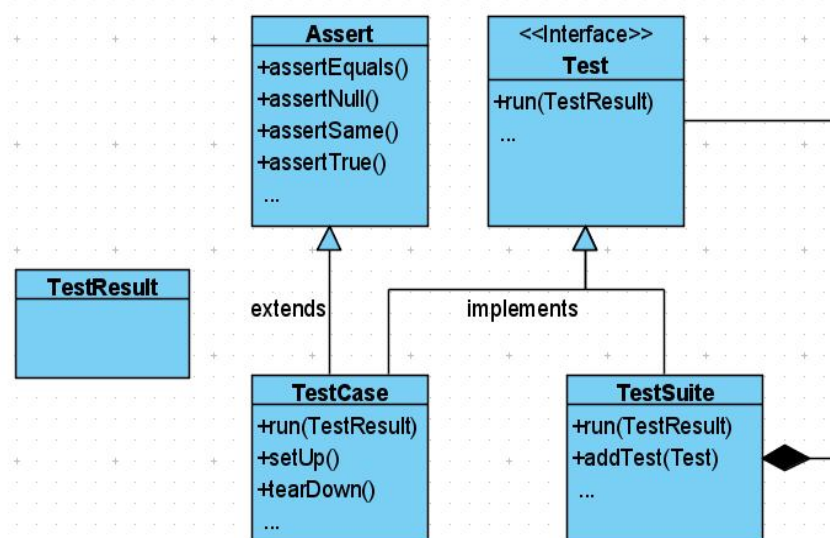
- Caractéristique d'un programme sans test:  
INNACHEVÉ

M.Romdhani, Avril 2017

39

## Architecture de JUnit

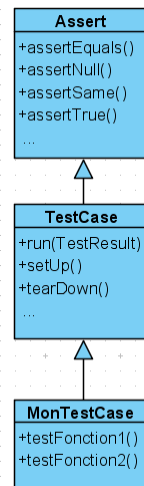
Amélioration de la qualité du code



M.Romdhani, Avril 2017

40

## Terminologie : les tests cases



### ■ Ecrire une classe de test consiste à:

- hériter de la classe TestCase
- implémenter plusieurs méthodes nommées test<f>()
- utiliser des assertions (assertXXX()):
  - assertTrue(1 > 0);
  - assertEquals(7,3+4);

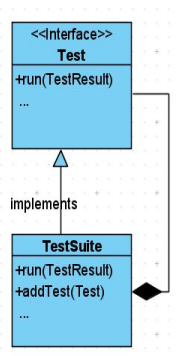
## Terminologie : les assertions

- **static void assertTrue(boolean condition)**
  - Vérifie si condition est vraie/fausse.
- **static void assertFalse(boolean condition)**
  - Vérifie si condition est vraie/fausse.
- **static void assertEquals(expected, actual)**
  - Vérifie que expected & actual sont égaux.
- **static void assertSame(expected, actual)**
  - Vérifie que expected & actual réfèrent au même objet.
- **static void assertNotSame(expected, actual)**
  - Vérifie que expected & actual réfèrent au même objet.
- **static void assertNull(Object objet)**
  - Vérifie que objet est null/pas null.
- **static void assertNotNull(Object objet)**
  - Vérifie que objet est null/pas null.
- **static void fail()**
  - Provoquer l'échec du test.

## Terminologie : les fixtures

- C'est le code de mise en place du contexte de test:
- Des sous-classes de `TestCase` qui ont plusieurs méthodes `testXXX()` peuvent utiliser les méthodes `setUp()` et `tearDown()` pour initialiser, resp. nettoyer, le contexte commun aux tests (= fixture)
  - Chaque test s'exécute dans le contexte de sa propre installation, en appelant `setUp()` avant et `tearDown()` après chaque méthode de test.
  - Pour deux méthodes, exécution équivalente à :
    - `setUp(); testMethod1(); tearDown();`
    - `setUp(); testMethod2(); tearDown();`
  - Le contexte est défini par des attributs du `TestCase`.

## Terminologie : les tests suites



- Une `TestSuite` peut être formée de `TestCase` ou de `TestSuite`.
- Elle appelle automatiquement toutes les méthodes `testXXX()` de chaque `TestCase`.
- 2 possibilités pour construire une `TestSuite`:
  - ✓ explicitement:
    - Ajouter les méthodes à tester en faisant appel à la méthode `addTest(Test test)`:  
`public void addTest(Test test)`
  - ✓ utilisant l'inspection  
`public TestSuite(java.lang.class testClass)`
    - Adds all the methods starting with "test" as test cases to the suite.

## Terminologie : les tests runners

Amélioration de la qualité du code

- Utilitaires pour lancer une suite de tests:
- On trouve les utilitaires JUnit
  - Textuel:
    - `junit.textui.TestRunner.run(TestClass.class);`
  - Graphique:
    - Awt:
      - `junit.awtui.TestRunner.run(TestClass.class);`
    - Swing:
      - `junit.swingui.TestRunner.run(TestClass.class);`
- On trouve les utilitaires liés aux plugins intégrés aux IDE :



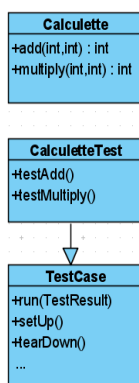
M.Romdhani, Avril 2017

45

## Ecriture d'un test (1)

Amélioration de la qualité du code

- Un « TestCase » avec « fixture »:



```

public class CalculetteTest extends TestCase {
    //déclaration fixture
    private Calculette _calc;

    public CalculetteTest(String name) {}

    protected void setUp() throws Exception {
        //initialisation fixture
        super.setUp();
        this._calc=new Calculette();
    }

    protected void tearDown() throws Exception {
        //libération fixture
        super.tearDown();
        this._calc=null;
    }
}
  
```

M.Romdhani, Avril 2017

46

Amélioration de la qualité du code

## Ecriture d'un test (2)

➤ Ecriture des méthodes de test:

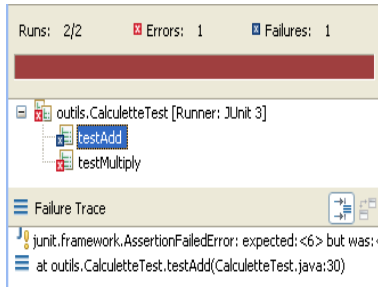
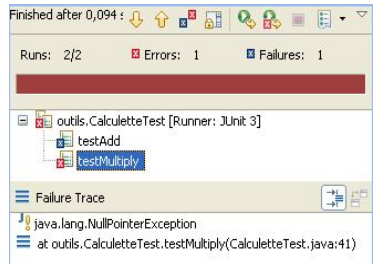
```
public void testAdd() {
    int expected = 6;
    int actual = this._calc.add(0, 4);
    this.assertEquals(expected, actual);

    //fail("Not yet implemented");
}
```

```
public void testMultiply() {
    //lever une exception "NullPointerException"
    String s=null;
    System.out.println(s.toString());

    //fail("Not yet implemented");
}
```

➤ Résultats:

**47**

M.Romdhani, Avril 2017

Amélioration de la qualité du code

## JUnit 4

- **Le framework JUnit a été largement remanié lors de son passage à la version 4**
  - Utilisation des annotations
  - Utilisation du framework Hamcrest
- **L'utilisation dans les projets nécessite l'importation des archives (répertoire plugins)**
  - junit.jar de JUnit 4
  - org.hamcrest.core\_1.1.0.v20090501071000.jar
- **Nouveautés de JUnit 4**
  - Les méthodes de test sont annotées avec @Test
  - pas besoin de dériver de TestCase
  - les assertions sont des méthodes statiques de la classe org.junit.Assert
  - les méthodes n'ont pas l'obligation d'être nommées en testXxxx
  - Paramètres principaux de l'annotation
    - expected : classe Throwable attendue
    - timeout : durée maximale en millisecondes
    - @Ignore permet d'ignorer un test

**48**

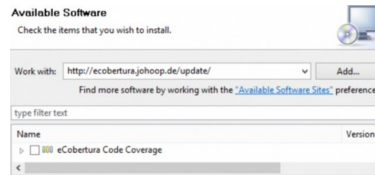
M.Romdhani, Avril 2017



## Code Coverage with Corbertura

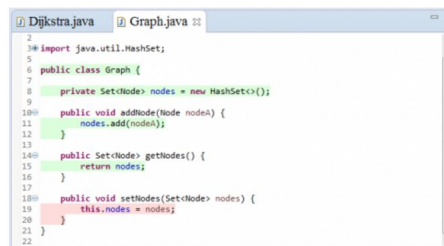
- Cobertura is a reporting tool that calculates test coverage for a codebase – the percentage of branches/lines accessed by unit tests in a Java project.

- Eclipse plugin :



- Code Coverage Reports in Eclipse :

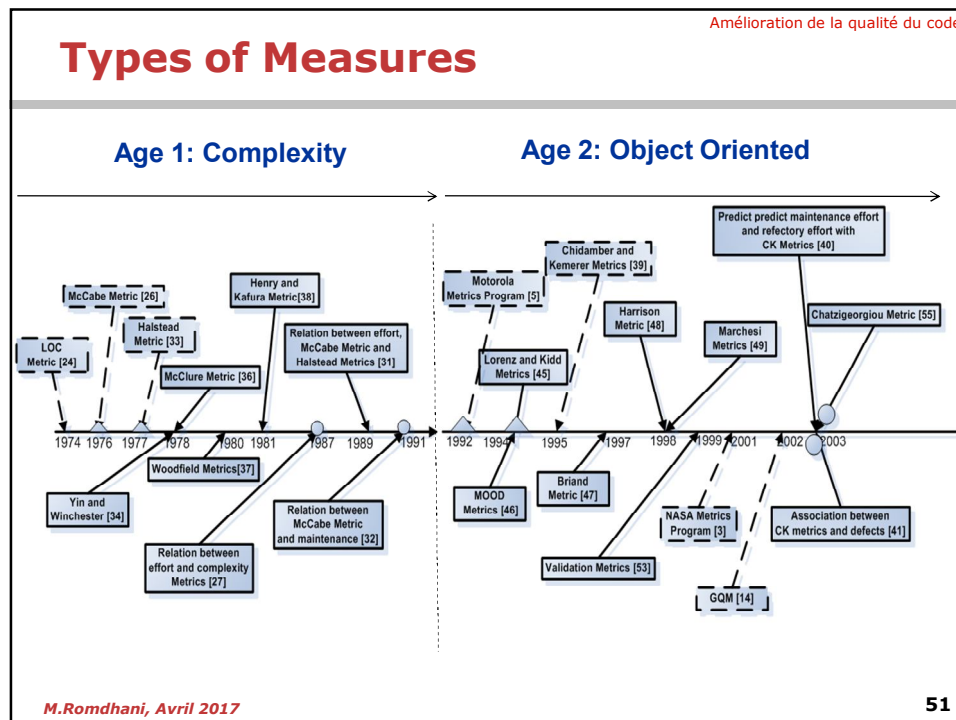
- In order to calculate code coverage by a Unit Test, right-click your project/test to open the context menu, then choose the option *Cover As* → *JUnit Test*.
- Under the *Coverage Session* view, you can check the line/branch coverage report per class



M.Romdhani, Avril 2017

49

## Métriques du logiciel



Amélioration de la qualité du code

## The CK Metrics Suite (1994)

- One of most referenced set of metrics
- Six metrics measuring class size and complexity, use of inheritance, coupling between classes, cohesion of a class and collaboration between classes

52

M.Romdhani, Avril 2017

## CK\_1: Weighted Methods per Class (WMC)

- Number of methods weighted by their procedural complexity (~complexity of class). Using unity weights gives simply number of methods in a class.

$$\text{WMC} = \sum_{i=1}^n C_i \quad C_i \text{ is the complexity (e.g., volume, cyclomatic complexity, etc.) of each method}$$

- WMC is indicator of the amount of effort required to implement and test a class.
- As the number of methods for a class grows, it is likely to become more application specific and thus limiting possibilities for reuse
- High WMC value means also greater potential impact on children
- Difficult to set exact limits for metric, but WMC should be kept as low as possible
- High WMC values -> split class

## CK\_2: Lack of Cohesion in Methods (LCOM)

- Cohesion measures "togetherness" of a class: high cohesion means good class subdivision (encapsulation)
- LCOM counts the sets of methods that are not related through the sharing of some of the class's instance variables
- LCOM\* normalized version, range of values between 0..1
- LCOM\* = 0 if every method uses all instance variables
- LCOM\* = 1 if every method uses only one instance variable
- High values of LCOM indicate scatter in the functionality provided by a class, i.e. class attempts to provide many different objectives
- Classes with high LCOM can be fault-prone and are likely to behave less predictable ways than classes with low LCOM
- High LCOM -> split class

## CK\_2: Lack of Cohesion in Methods (LCOM)

Amélioration de la qualité du code

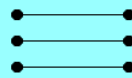
### ■ Problems with LCOM:

- Gives equal values for very different classes
- Classes with "getters & setters" (getProperty(), setProperty()) get high LCOM values although this is not an indication of a problem
- Don't even try to measure logical cohesion

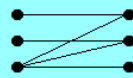
$$LCOM^* = \frac{1/a * (\sum_j (\#methods \text{ using attribute } j)) - n}{1 - n}$$

a = # instance variables  
n = # methods  
j = 1..a

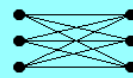
Methods on the left, instance variables on the right



LCOM\* = 1



LCOM\* = 0.67



LCOM\* = 0

M.Romdhani, Avril 2017

55

## LCOM

Amélioration de la qualité du code

- Class  $C_k$  with  $n$  methods  $M_1, \dots, M_n$
- $I_j$  is the set of instance variables used by  $M_j$
- There are  $n$  such sets  $I_1, \dots, I_n$ 
  - $P = \{(I_i, I_j) \mid (I_i \cap I_j) = \emptyset\}$
  - $Q = \{(I_i, I_j) \mid (I_i \cap I_j) \neq \emptyset\}$
- If all  $n$  sets  $I_i$  are  $\emptyset$  then  $P = \emptyset$
- $LCOM = |P| - |Q|$ , if  $|P| > |Q|$
- $LCOM = 0$  otherwise

M.Romdhani, Avril 2017

56

### CK\_3: Response For a Class (RFC)

- Number of methods that can be invoked in response to a message
- Measure of potential communication between classes
- Often computed simply by counting number of method calls at class's method bodies (computing full transitive closure of each method is slow)
- Complexity of the class increases and understandability decreases as RFC grows
- Testing gets harder as RFC grows (better understanding required from a tester)
- RFC can be used as indicator of required testing time
- High RFC -> there could be a better class subdivision (e.g. merge classes)

### CK\_4: Depth of Inheritance Tree (DIT)

- Maximum height of inheritance tree or level of a particular class in a tree
- As DIT grows, it is likely that classes on lower level inherits lots of methods and overrides some. Thus predicting behavior for an object of a class becomes difficult.
- Large DIT means greater design complexity
- DIT also measures reuse via inheritance
- High or low values of DIT might indicate problems with domain analysis

## CK\_5: Number Of Children (NOC)

- Number of immediate subclasses for a class
- Indicator of potential influence that class has on design
- High value of NOC might indicate misuse of subclassing (=implementation inheritance instead of is-a relationship)
- Class with very high NOC might be a candidate for refactoring to create more maintainable hierarchy
- NOC is also measure of reuse via inheritance
- NOC can be used as indicator of required testing time

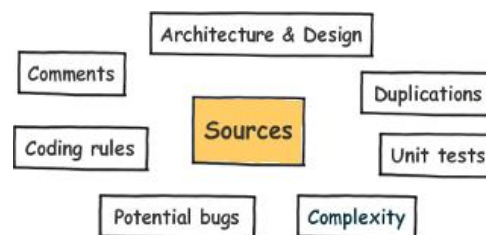
## CK\_6: Coupling Between Objects (CBO)

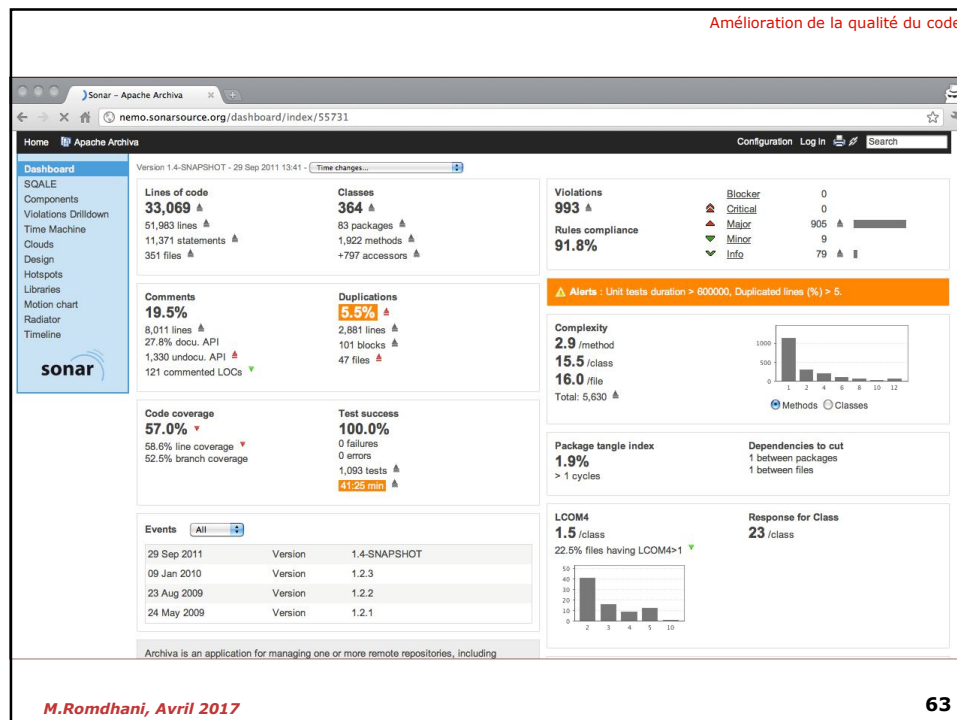
- Coupling = class x is coupled to class y iff x uses y's methods or instance variables (includes inheritance related coupling)
- CBO for a class is a count of the number of other classes to which it is coupled
- High coupling between classes means modules depend on each other too much
- CBO is the number of collaborations between two classes (fan-out of a class C)
  - The number of other classes that are referenced in the class C (a reference to another class, A, is an reference to a method or a data member of class A)
  - Viewpoints:
    - As collaboration increases reuse decreases
    - High fan-outs represent class coupling to other classes/objects and thus are undesirable
    - High fan-ins represent good object designs and high level of reuse
    - Not possible to maintain high fan-in and low fan outs across the entire system
- Independent classes are easier to reuse and extend
- High coupling decreases understandability and increases complexity
- High coupling makes maintenance more difficult since changes in a class might propagate to other parts of software
- Coupling should be kept low, but some coupling is necessary for a functional system

## Indicateurs de qualité avec SonarQube

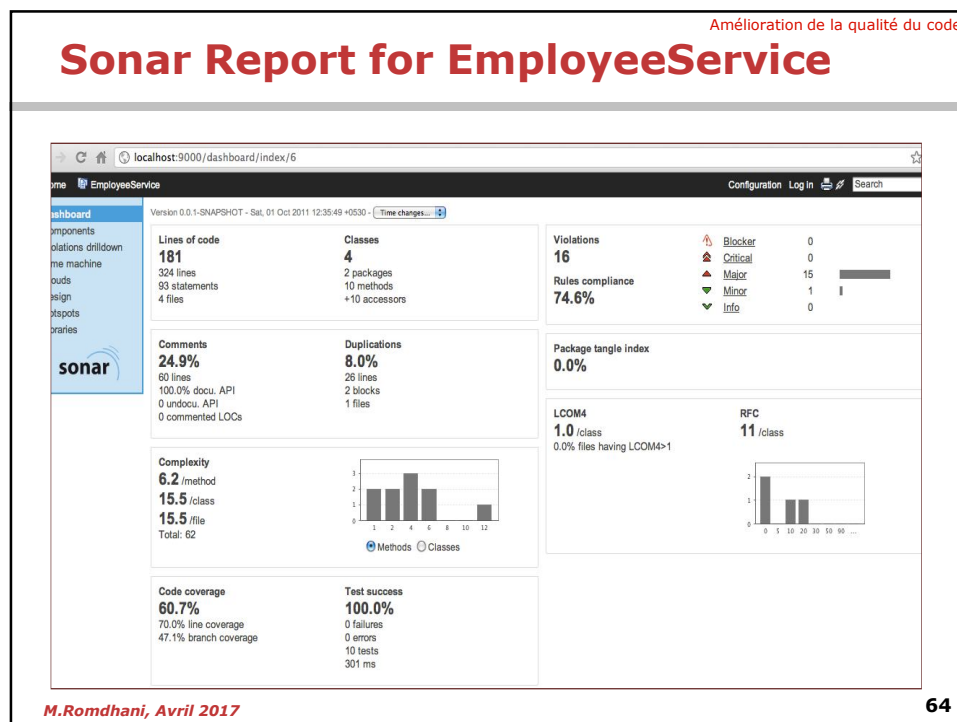
## What is Sonar ?

- Sonar is a code quality tool





M.Romdhani, Avril 2017



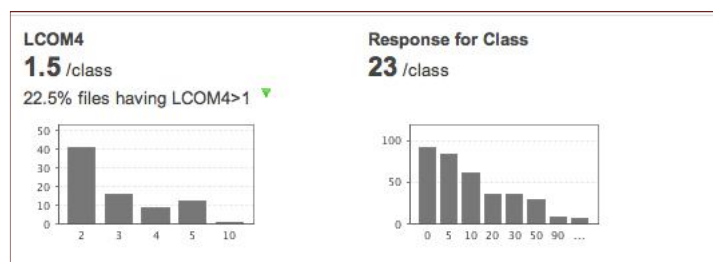
M.Romdhani, Avril 2017



## Basic Metrics (Starter pack)

- Lines of Code/Classes/Methods
- Rules Compliance Index & Violations
- Comments and Duplicate Code
- Package Tangle Index
- Method/Class Complexity (Cyclometric)
- LCOM4 and RFC
- Code Coverage and Test Results

## LCOM4 and RFC



## LCOM4

Amélioration de la qualité du code

- Lets Start with SOLID Design Principle
  - S = Single Responsibility Principle
- A Class should have only one responsibility
- If Class has more than one
  - Then break the class into smaller classes
- This ensures
  - Modularity
  - Reusability

M.Romdhani, Avril 2017

67

## How to measure LCOM4

Amélioration de la qualité du code

- If a class as 2+ sets of method totally disjoint, then we can very much say class has 2 responsibility
- <http://www.sonarsource.org/clean-up-design-at-class-level-with-sonar/>

M.Romdhani, Avril 2017

68

## RFC – Response for Class

- Total number of methods/constructor invoked as a result of calling the method of a class

## Code Coverage and Test Results

<b>Code coverage</b>	<b>Test success</b>
<b>57.0%</b> ▼	<b>100.0%</b>
58.6% line coverage ▼	0 failures
52.5% branch coverage	0 errors
	1,093 tests ▲
	<b>41:25 min</b> ▲

## Code Coverage and Test Results

- Code Coverage is the paths of code covered by unit test
- Test Results is how many test cases passed or fail