Qualité du logiciel ?

Merci à tous ceux qui ont rendu leurs cours et exposés
disponibles sur le web & dans les livres,
voir Biblio.

M. Blay-Fornarino
blay@unice.fr,
http://users.polytech.unice.fr/~blay/
IUT Département Informatique 2è année

Bibliographie

- Reflexion on Software Quality and Maintenance, Alexandre Bergel, Chili
- Cours de Production Du Logiciel, La qualité logicielle, Thierry Milan, Toulouse
- Yann-Gaël Guéhéneuc cours , Université de Montréal, http://www-etud.iro.umontreal.ca/~ptidej/yann-gael/Work/Publications/

Le logiciel ...

S'il vous fallait préparer à l'avance tous les ordres à donner à des individus totalement stupides, mais absolument obéissants, pour qu'ils réalisent une tâche complexe, vous diriez que c'est là un problème de management nouveau : les gens ne sont, en général, ni stupides, ni obéissants, ce qui aide à faire face à l'imprévu. Si, en plus, la moindre erreur provoque une catastrophe, vous diriez que la réussite d'un tel plan d'action tient du miracle. Vous venez pourtant de définir ce qu'est la fabrication d'un logiciel. Pour que la réussite ne tienne pas que du miracle, il faut donc une rigueur de fer, pas mal de culture, et de bons outils tels que ceux qui apparaissent aujourd'hui, ce qui n'est malheureusement pas encore assez connu.

LOGICIELS : COMMENT CHASSER LES BUGS ?

par

Gérard BERRY 1996

Directeur de Centre de Mathématiques Appliquées
de l'École des mines de Paris

membre de l'Académie des sciences française (depuis 2002),
de l'Académie des technologies (depuis 2005),
et de l'Academia Europaea (depuis 1998)

Deux points de vue sur la qualité du code

Est-ce que le logiciel met en œuvre correctement les exigences?
Est-il facile à utiliser? Plante-il?

Comment est organisé le code?
Où est implémentée cette fonctionnalité ?

External (user's): *what?*

Internal (programmer's): *how?*

indico.cern.ch/event/34666/session/11/contribution/13/material/.../0.ppt

User's frustration

- "after clicking all options and navigating all menus I still cannot find out how to achieve <this>"
- "why do I get this error message? I did nothing wrong – stupid software!"

Programmer's frustration

- "I have to add this function, but it's impossible to find a single place where the addition should go in the intricate code structure"
- "the program produces the wrong output, but after several hours browsing the code I cannot identify which statements are guilty"

Programmer's frustration

- Pour un développeur, la qualité d'une application réside dans la capacité de son code à être lu, compris et repris par d'autres développeurs. Un code de mauvaise qualité sera plus difficile à maintenir et source d'anomalies.
- L'incompréhension est, en effet, une cause d'erreurs non négligeable mais peut aussi devenir une surcharge importante pour le projet.

Qualité ?

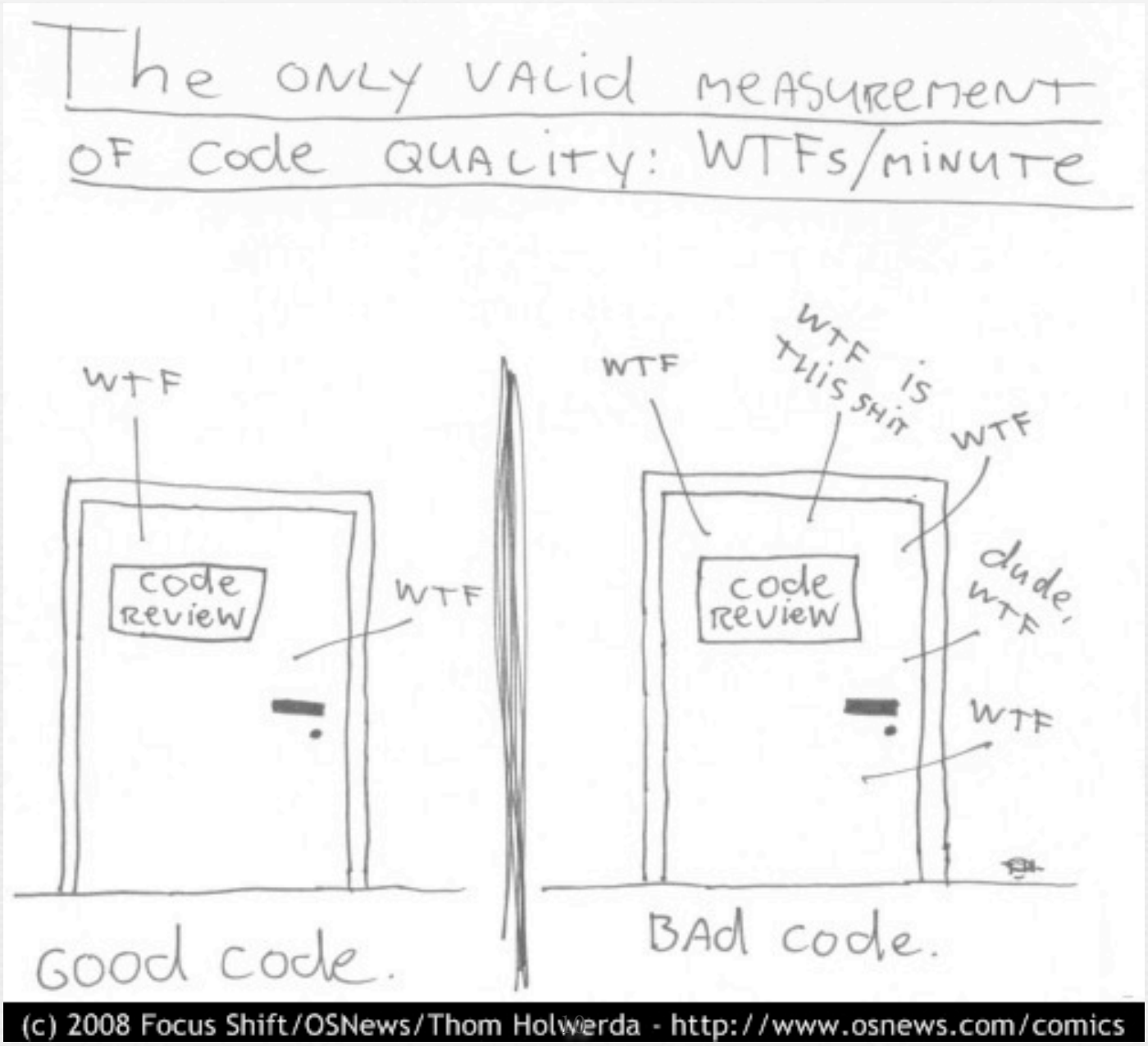
- Exemple
- On compare deux logiciels
  - Le premier a encore 10 erreurs résiduelles
  - Le second n'a que trois erreurs résiduelles
- Est-ce que le second est de meilleure qualité que le premier?

- La qualité pour n'importe quel processus est difficile à définir
- Il faut identifier
  - Des facteurs
  - Des critères
  - Des métriques

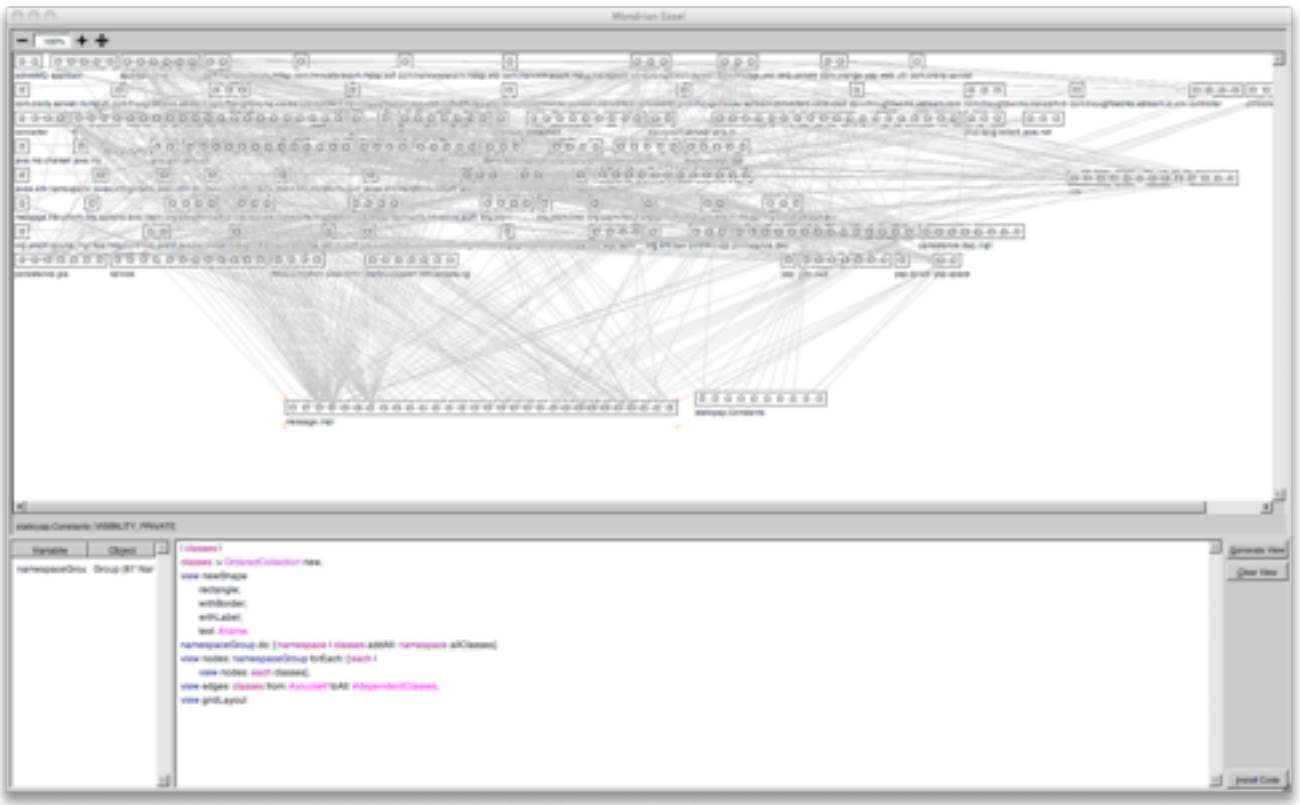
Michel Koenig,
Polytech'Nice Sophia

Evaluation de la Qualité ?

Evaluation


The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE
WTF
code review
WTF
Good code.
WTF
WTF is THIS SHIT
WTF
code review
dude, WTF
WTF
BAd code.
(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics

Large software in a French telecom company



~100 packages
~ 500 classes

Paris, 2008

Intuitivement, la qualité ...

à partir du cours d'Alexandre Bergel


```
/* Hello World program */

#include<stdio.h>

int main()
{
  printf("Hello World");

}
```

no return statement in the function

```
/* Hello World program */

#include<stdio.h>

int main()
{
  printf("Hello World");

}
```

no return statement in the function

```
/* Hello World program */

#include<stdio.h>

int main()
{
  printf("Hello World");

}
```

9 lines of code
7 seconds

In Swing:

JComponent.java

```
protected String getBorderTitle(Border b) {
  String s;
  if (b instanceof TitledBorder) {
       return ((TitledBorder) b).getTitle();
  } else if (b instanceof CompoundBorder) {
       s = getBorderTitle(((CompoundBorder)
                          b).getInsideBorder());
  if (s == null) {
    s = getBorderTitle(((CompoundBorder)
                          b).getOutsideBorder());
  }
  return s;
   } else {
      return null;
   }
...
```

In Swing:

JComponent.java

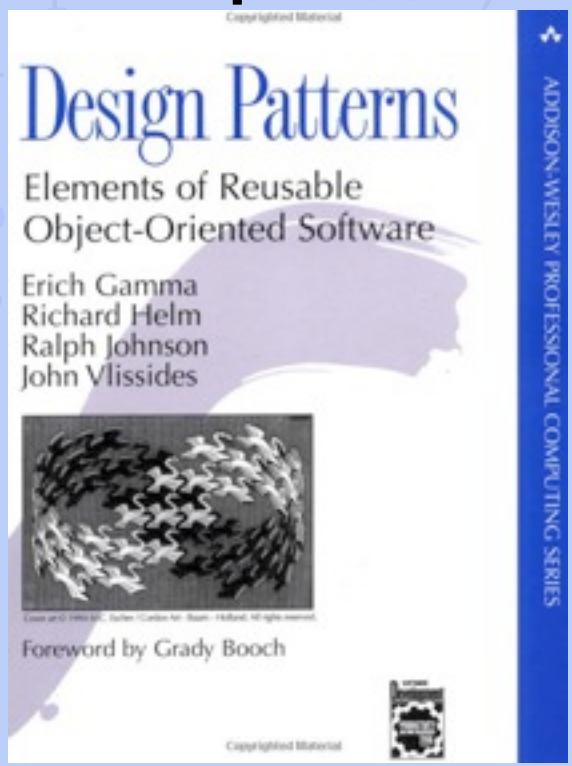
```
protected String getBorderTitle(Border b) {
   String s;
   if (b instanceof TitledBorder) {
         return ((TitledBorder) b).getTitle();
   } else if (b instanceof CompoundBorder) {
         s = getBorderTitle(((CompoundBorder)
                        b).getInsideBorder());
   if (s == null) {
     s = getBorderTitle(((CompoundBorder)
                        b).getOutsideBorder());
   }
   return s;
    } else {
       return null;
    }
...
```

In Swing:

JComponent.java

```
protected String getBorderTitle(Border b) {
   String s;
   if (b instanceof TitledBorder) {
         return ((TitledBorder) b).getTitle();
   } else if (b instanceof C
         s = getBorderTitle((
                        b).getI
   if (s == null) {
     s = getBorderTitle(((Co
                        b).getO
   }
   return s;
    } else {
       return null;
    }
...
```

Need Double Dispatch



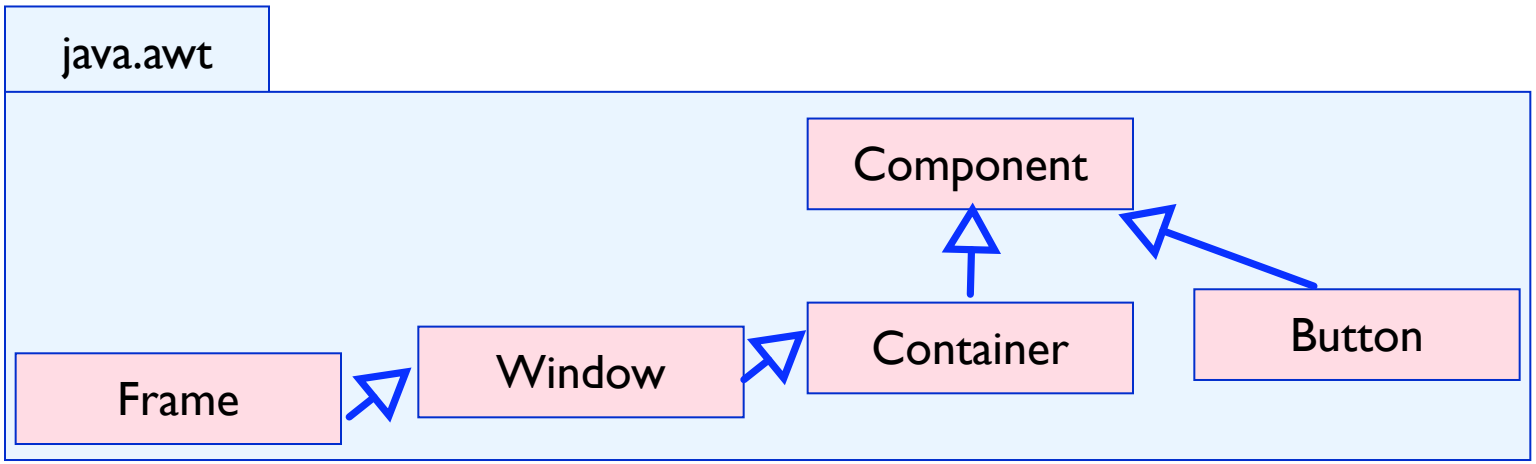
In Swing:

JComponent.java

```
protected String getBorderTitle(Border b) {
   String s;
   if (b instanceof TitledBorder) {
         return ((TitledBorder) b).getTitle();
   } else if (b instanceof CompoundBorder) {
         s = getBorderTitle(((CompoundBorder)
                  b).getInsideBorder());

   if (s == null) {
     s = getBorderTitle(((CompoundBorder)
                  b).getOutsideBorder());

   }
   return s;
    } else {
       return null;
    }
...
```

5471 lines of code
12 minutes

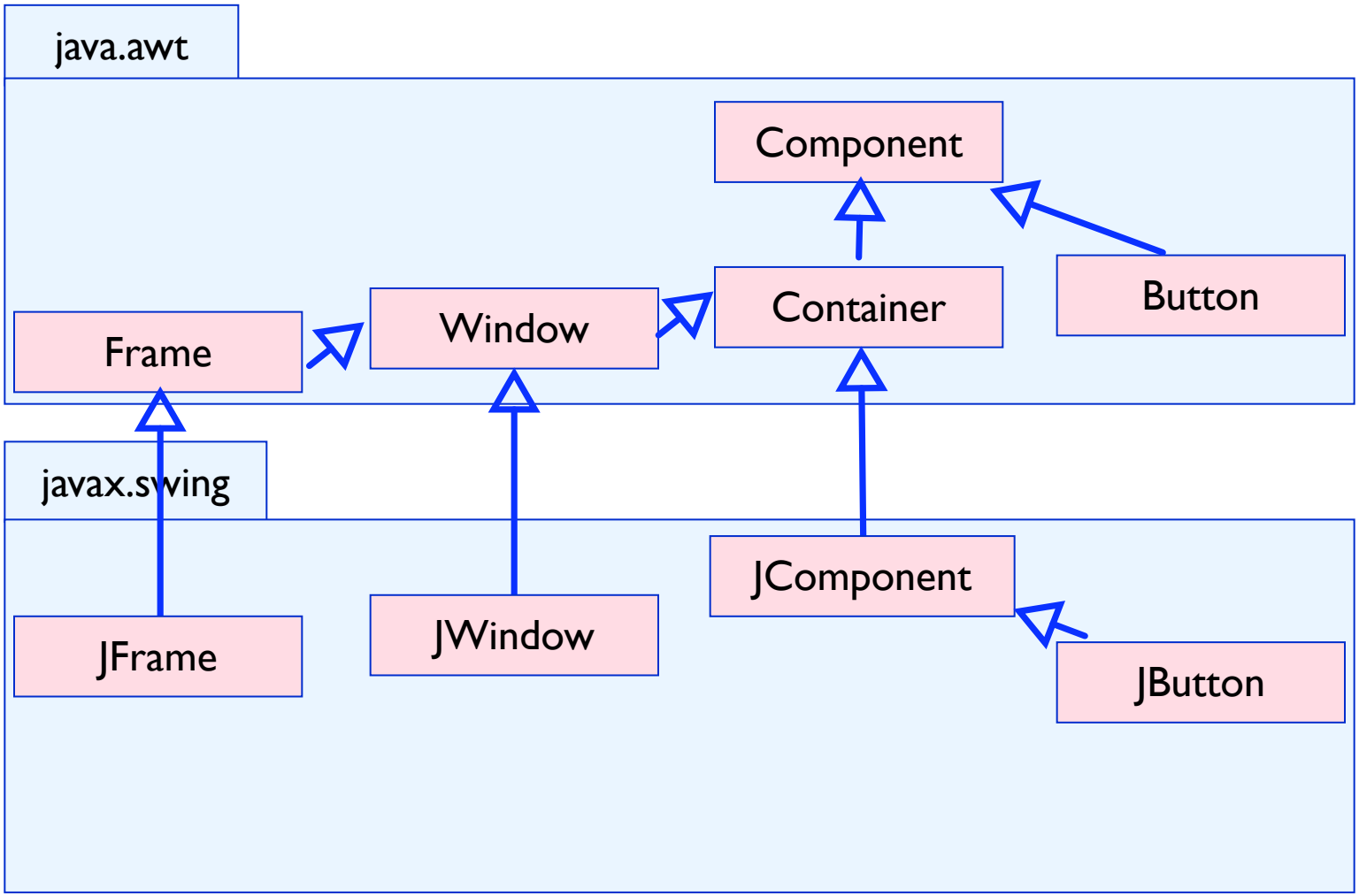
Presentation of AWT



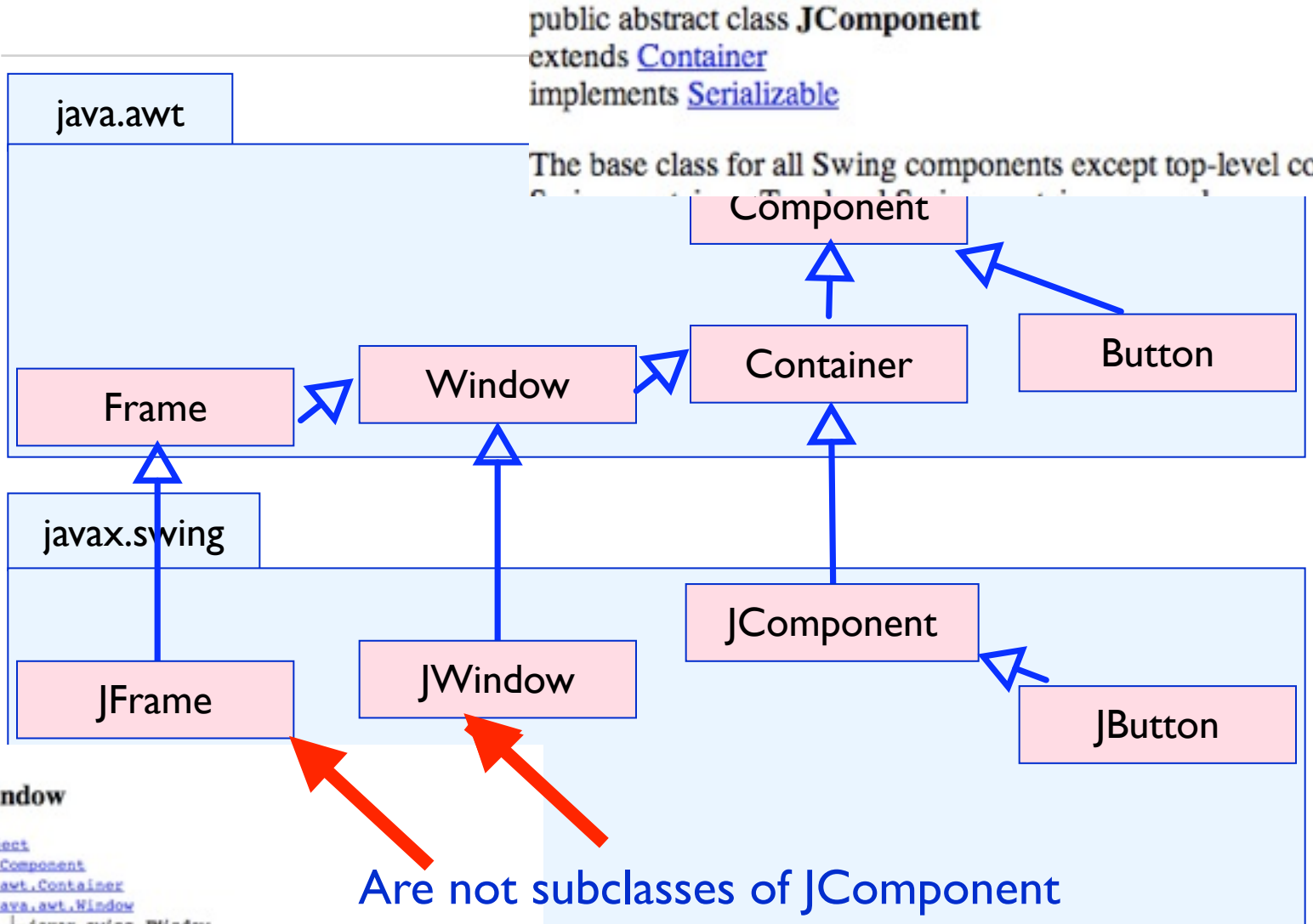
In the AWT framework:

- Widgets are components (i.e., inherit from Component)
- A frame is a window (Frame is a subclass of Window)

Swing at the top of AWT



Problem #1: Brocken Inheritance



public abstract class **JComponent**
extends Container
implements Serializable

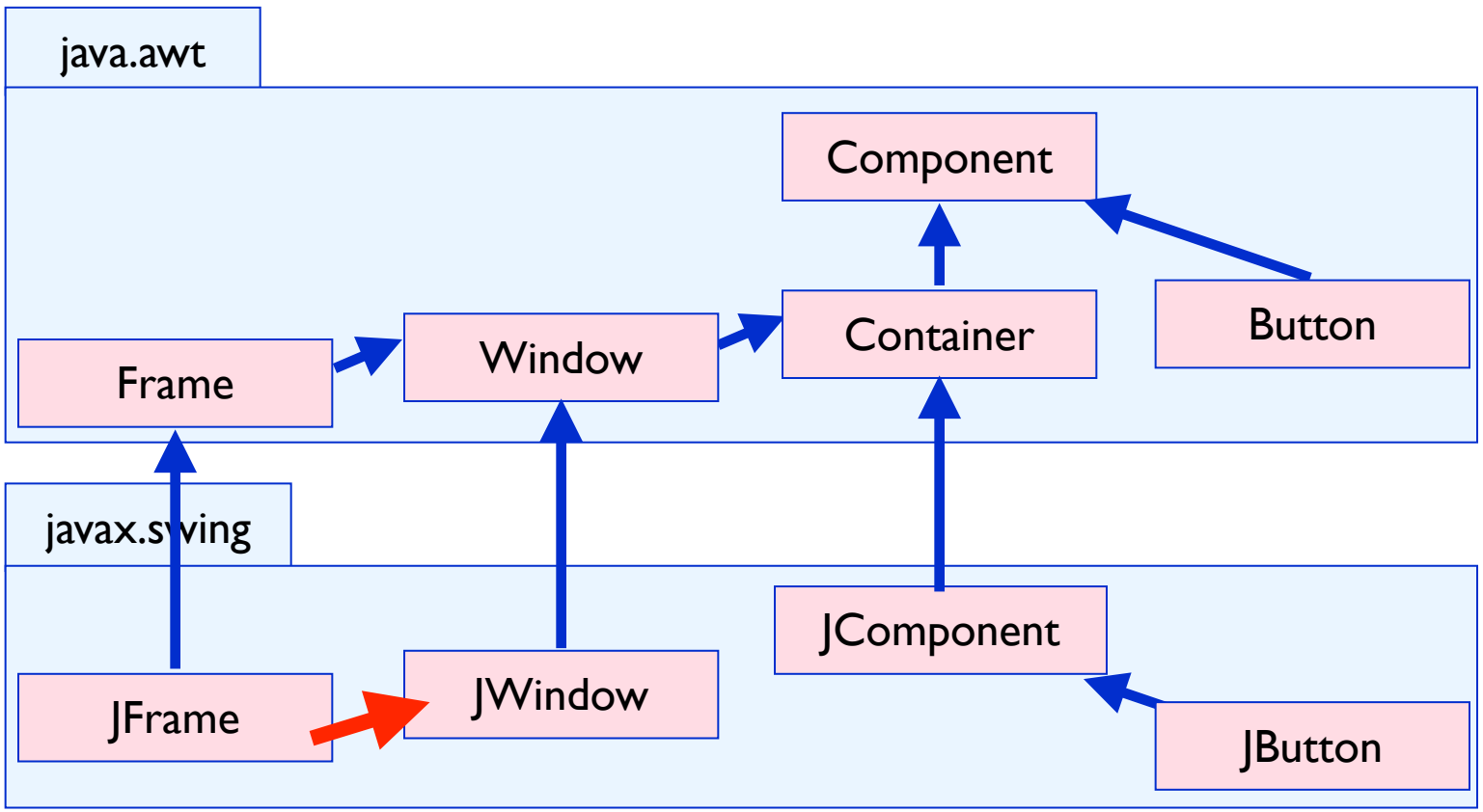
The base class for all Swing components except top-level containers.

javax.swing
Class JWindow

java.lang.Object
└ java.awt.Component
└ java.awt.Container
└ java.awt.Window
└ javax.swing.JWindow

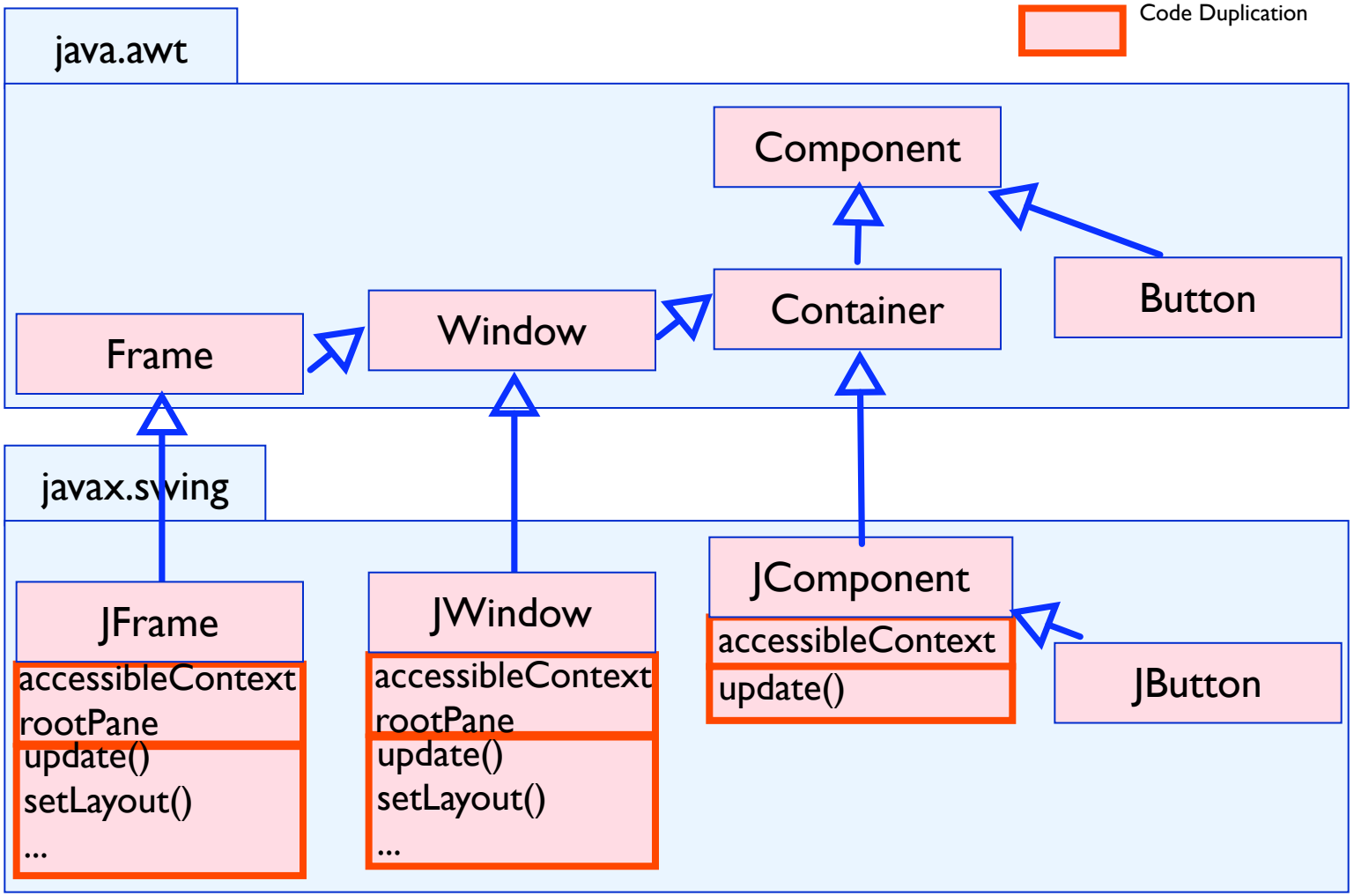
All Implemented Interfaces:
Accessible, ImageObserver, MenuContainer, RootPaneContainer, Serializable

Problem #1: Brocken Inheritance



Missing inheritance link between JFrame and JWindow

Problem #2: Code Duplication



Problem #3: Explicit Type Checks and Casts

```
public class Container extends Component {
  Component components[] = new Component [0];
  public Component add (Component comp) {...}
}

public class JComponent extends Container {
  public void paintChildren (Graphics g) {
    for (; i>=0 ; i--) {
      Component comp = getComponent (i);
      isJComponent = (comp instanceof JComponent);
      ...
      ((JComponent) comp).getBounds();
    }
  }}
```

Supporting Unanticipated Changes

AWT couldn't *be enhanced* without risk of *breaking* existing code

Swing is, therefore, *built on the top* of AWT using *subclassing*

As a result, *Swing is a big mess internally*!

Why do we care to have a messy Swing ?

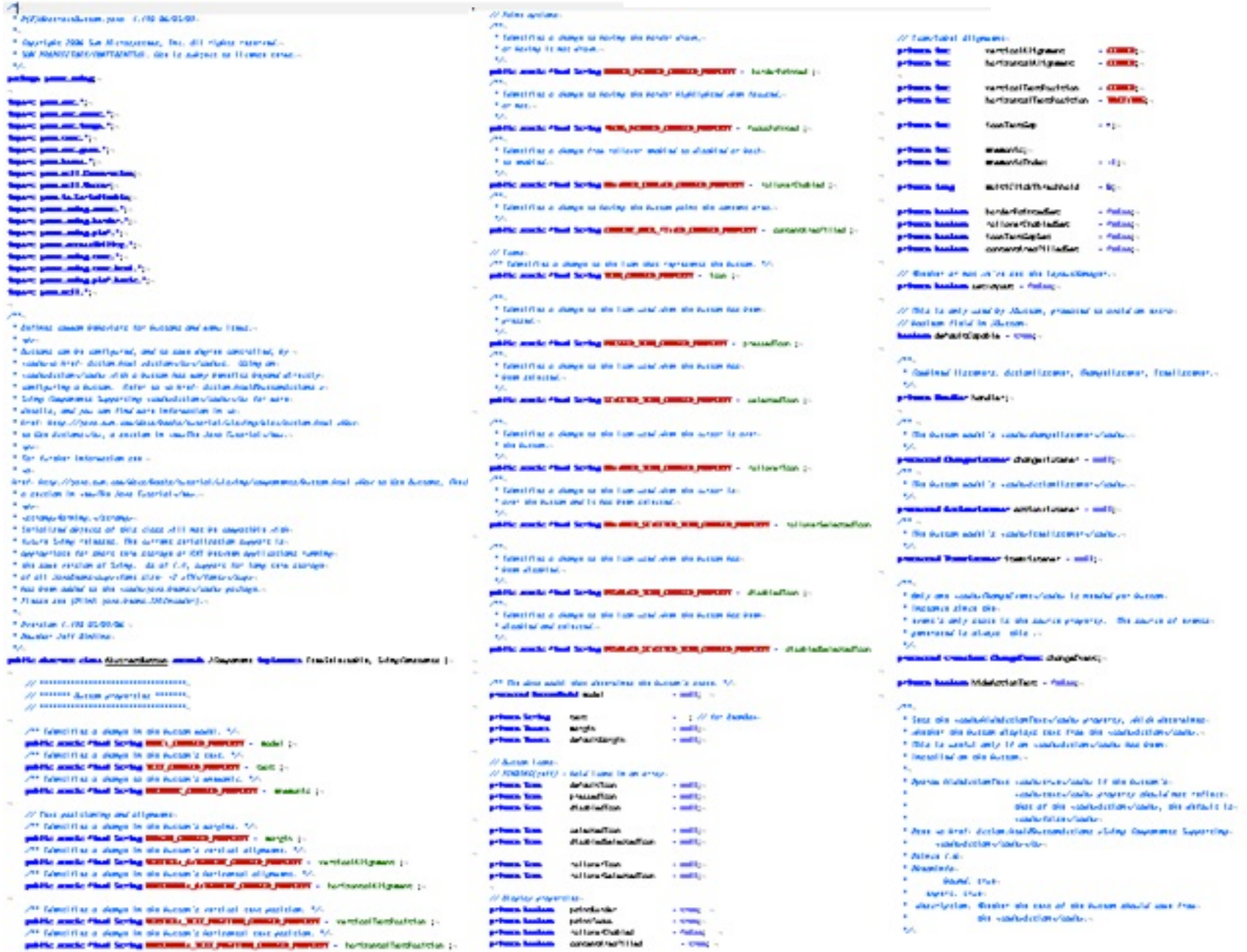
Swing appeared in 1998, and *has not evolved since*!

Swing is too heavy to be ported to PDA, cellphones, ...

SWT* is *becoming* a *new standard*.

Either a system evolves, or it is dead. [Lehmans74]

***SWT** is an open source widget toolkit for Java designed to provide efficient, portable access to the user-interface facilities of the operating systems on which it is implemented.



J2EE
3,430,663 lines of code
? hours

Linux 2.6.28
7,106,111 lines of code
? hours

Linux 2.6.28
7,106,111 lines of code
? hours

Qu'est-ce que la qualité ?

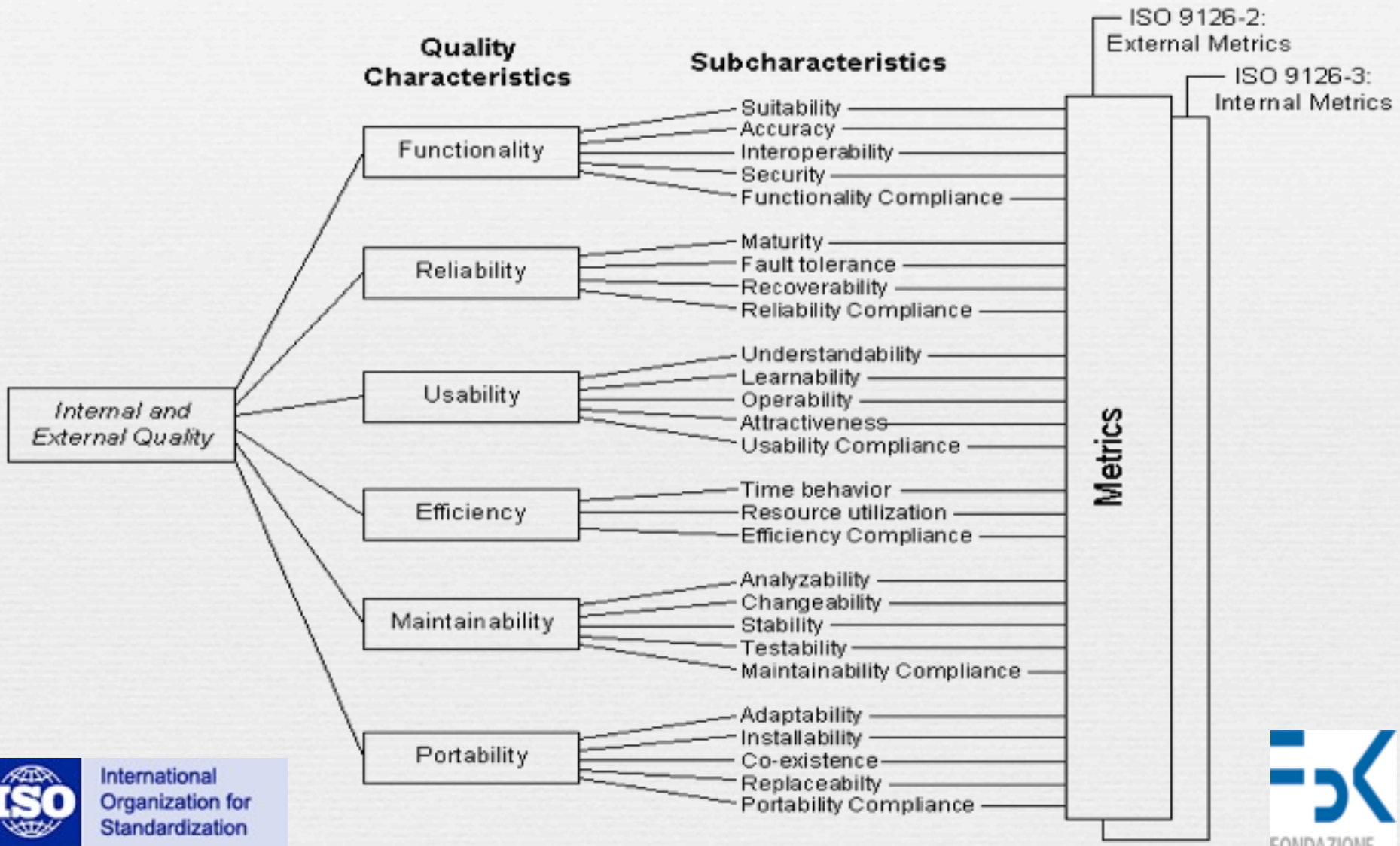
appréciation globale d'un logiciel,

basée sur de nombreux indicateurs

- ISO/CEI 9126 : **vocabulaire** visant à classifier l'ensemble des attributs d'un logiciel relevant de la qualité

ISO/IEC 9126-1,[1] classifies software quality in a structured set of characteristics and sub-characteristics.

Quality models (ISO 9126)

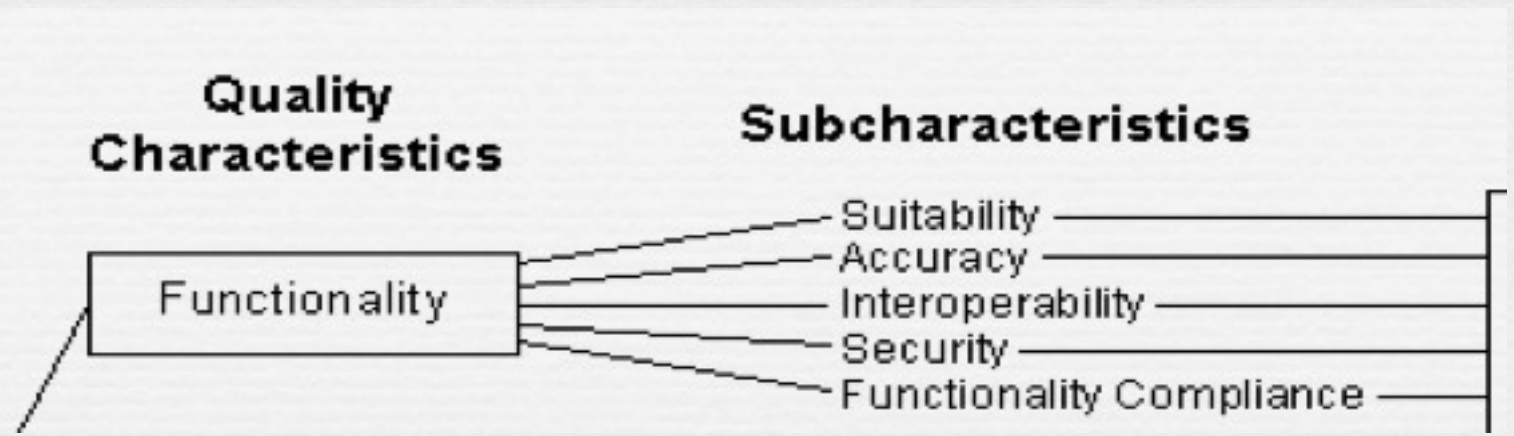


ISO International Organization for Standardization

FONDAZIONE BRUNO KESSLER

ISO/CEI 9126

- **Capacité fonctionnelle** (*functionality*)
  - ✓ Répond-il aux besoins de ses utilisateurs ?



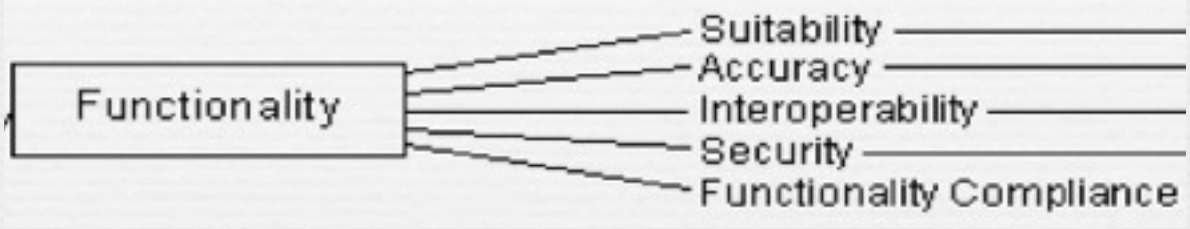
Capacité fonctionnelle

- Définition

– Ensemble d'attributs portant sur l'existence de fonctions et leurs propriétés; les fonctions sont celles qui satisfont aux besoins exprimés ou implicites

➡ Sous-caractéristiques

- **Aptitude** : présence et adéquation d'une série de fonctions pour les tâches données
- **Exactitude** : résultats ou effets justes ou convenus
- **Interopérabilité** : interactions avec d'autres systèmes
- **Sécurité** : accès non autorisé (accidentel ou délibéré) aux programmes et données



ISO/CEI 9126

- **Capacité fonctionnelle** (*functionality*)
  - ✓ Répond-il aux besoins de ses utilisateurs ?
- **Fiabilité** (*reliability*)
  - ✓ Est-il en mesure d'assurer un niveau de qualité de service suffisant pour satisfaire les besoins opérationnels de ses utilisateurs ?

Fiabilité

- Définition

– Ensemble d'attributs portant sur l'aptitude du logiciel à maintenir son niveau de service dans des conditions précises et pendant une période déterminée

➡ Sous-caractéristiques

– **Maturité** : fréquence des défaillances dues à des défauts
– **Tolérance aux fautes** : aptitude à maintenir un niveau de service donné en cas de défaut ou d'attaque
– **Possibilité de récupération** : capacité à rétablir son niveau de service et de restaurer les données directement affectées en cas de défaillance ; temps et effort nécessaire pour le faire

ISO/CEI 9126

- **Capacité fonctionnelle** (*functionality*)
  - ✓ Répond-il aux besoins de ses utilisateurs ?
- **Fiabilité** (*reliability*)
  - ✓ Est-il en mesure d'assurer un niveau de qualité de service suffisant pour satisfaire les besoins opérationnels de ses utilisateurs ?
- **Maintenabilité** (*maintainability*)
  - ✓ Est-il facile d'adapter le logiciel à de nouveaux besoins ou à de nouvelles contraintes ?

Maintenabilité

- Définition

– Ensemble d'attributs portant sur l'effort nécessaire pour faire des modifications données

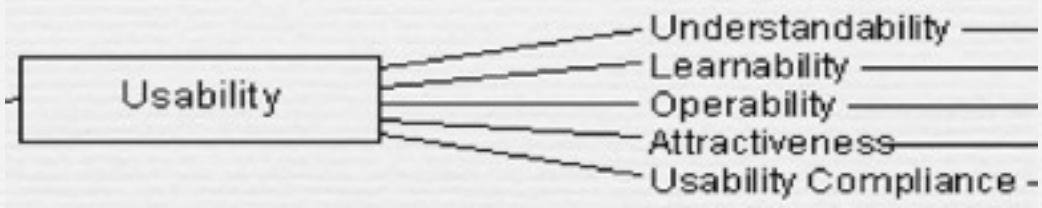
➡ Sous-caractéristiques

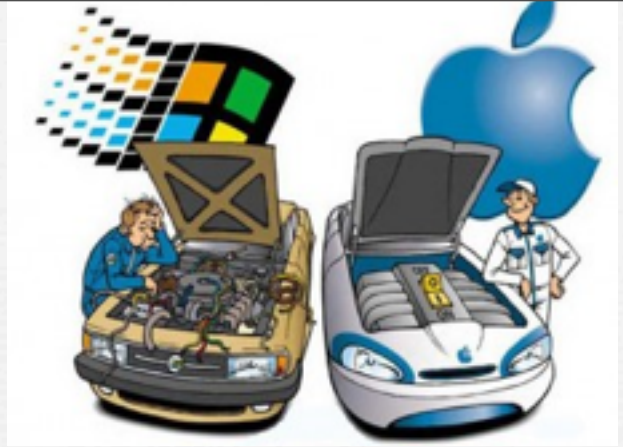
–**Facilité d'analyse** : effort nécessaire pour diagnostiquer les déficiences et leurs causes ou pour identifier les parties à modifier
–**Facilité de modification** : effort nécessaire pour modifier, remédier aux défauts ou adapter à l'environnement
–**Stabilité** : risque des effets inattendus des modifications
–**Facilité de test** : effort pour valider le logiciel modifié

ISO/CEI 9126

- **Facilité d'utilisation** (*usability*)
  - ✓ Peut-il être utilisé à moindre effort ?

Facilité d'utilisation





- Définition

– Ensemble d'attributs portant sur l'effort nécessaire pour l'utilisation et l'évaluation individuelle de cette utilisation par un ensemble défini ou implicite d'utilisateurs

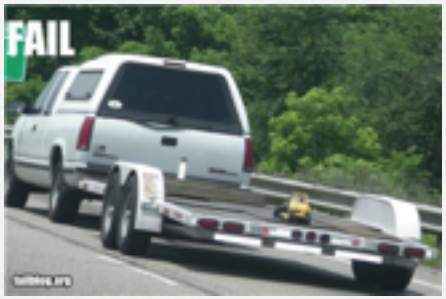
➡ Sous-caractéristiques

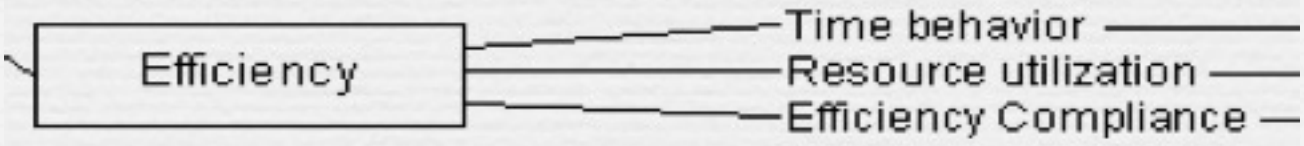
–**Facilité de compréhension** : effort de l'utilisateur pour comprendre la logique et la mise en œuvre
–**Facilité d'apprentissage** : effort de l'utilisateur pour apprendre son utilisation
–**Facilité d'exploitation** : effort que doit faire l'utilisateur pour exploiter et contrôler l'exploitation du logiciel

ISO/CEI 9126

- **Facilité d'utilisation** (*usability*)
  - ✓ Peut-il être utilisé à moindre effort ?
- **Rendement / Scalabilité** (*efficiency*)
  - ✓ Les ressources matérielles nécessaires à l'exécution du logiciel sont-elles en rapport avec sa rentabilité ?

Rendement





Définition

– Ensemble d'attributs portant sur le rapport existant entre le niveau de service d'un logiciel et la quantité de ressources utilisées, dans des conditions déterminées

➡ Sous-caractéristiques

–**Temps** : temps de réponses et de traitement ; débits lors de l'exécution de sa fonction
–**Ressources** : quantité de ressources utilisées ; durée de leur utilisation par fonction

ISO/CEI 9126

- **Facilité d'utilisation** (*usability*)
  - ✓ Peut-il être utilisé à moindre effort ?
- **Rendement / Scalabilité** (*efficiency*)
  - ✓ Les ressources matérielles nécessaires à l'exécution du logiciel sont-elles en rapport avec sa rentabilité ?
- **Portabilité** (*portability*)
  - ✓ Peut-il être transféré facilement d'une plate-forme ou d'un environnement à un autre ?

Portabilité

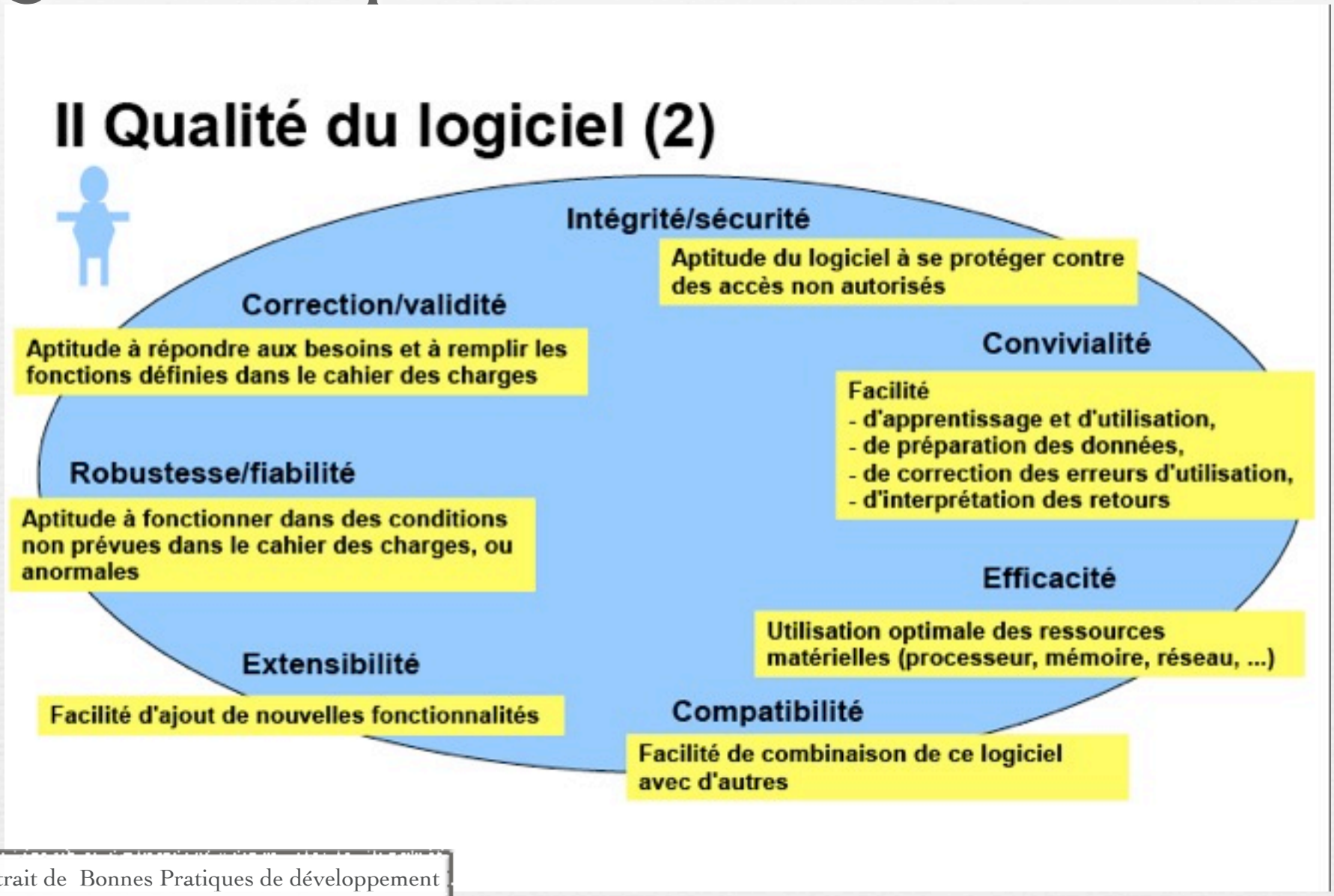
- Définitio..

– Ensemble d'attributs portant sur l'aptitude du logiciel à être transféré d'un environnement à un autre

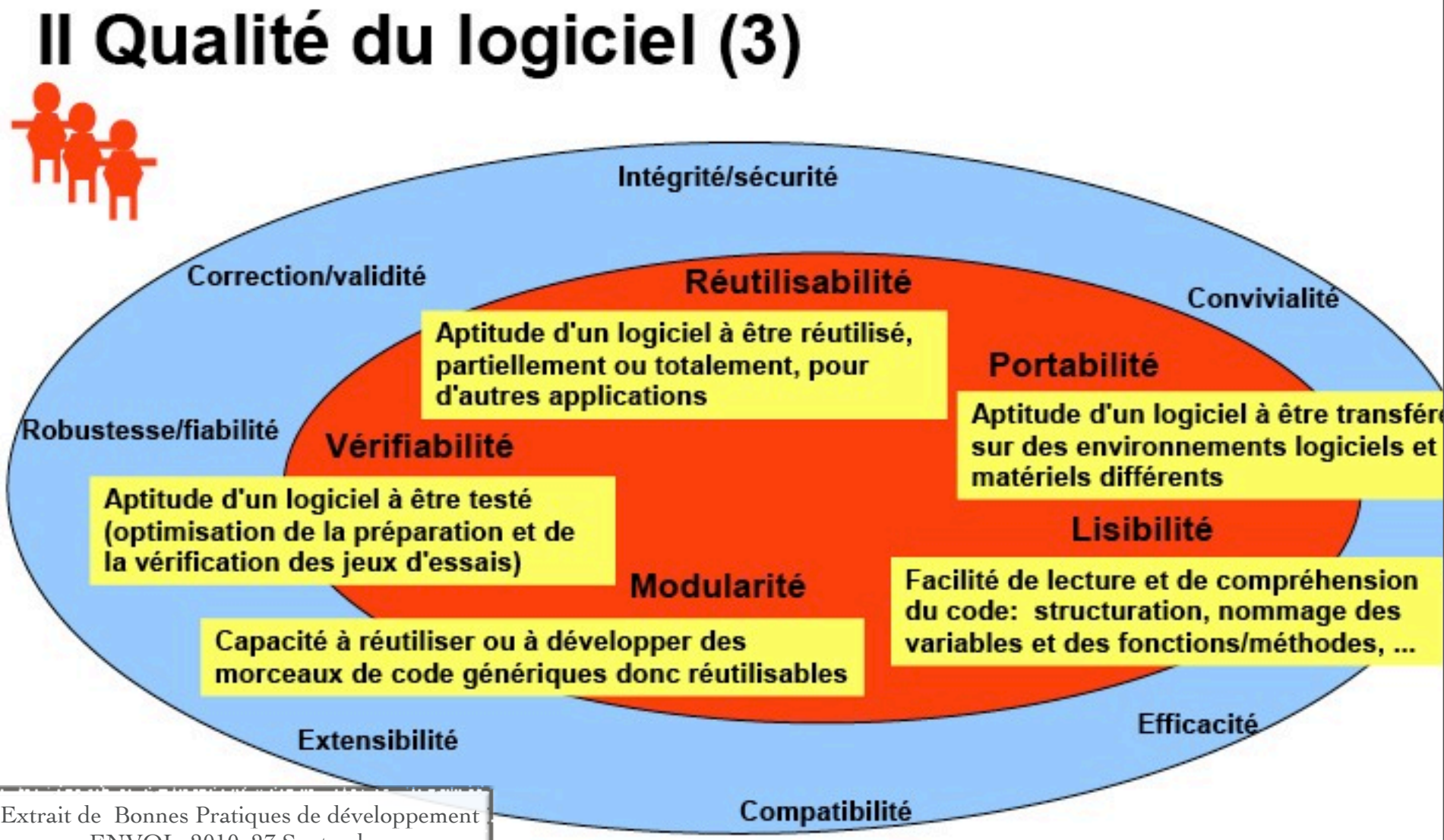
➡ Sous-caractéristiques

– **Facilité d'adaptation** : possibilité d'adaptation à différents environnements donnés sans que l'on ait recours à d'autres actions ou moyens que ceux prévus à cet effet par le logiciel.
– **Facilité d'installation** : effort nécessaire pour installer le logiciel dans un environnement donné.
– **Conformité aux règles de portabilité** : conformité aux normes et aux conventions ayant trait à la portabilité.
– **Interchangeabilité** : possibilité et effort d'utilisation du logiciel à la place d'un autre logiciel donné dans le même environnement.

Qualité : point de vue Utilisateur



Qualité : point de vue Concepteur



Extrait de Bonnes Pratiques de développement
ENVOL_2010: 27 Septembre
Véronique Baudin

Comment assurer la qualité ?

- **Les tests automatisés**
  - *Capacité fonctionnelle & Rendement*
    - Tests fonctionnels
    - Tests de charge
  - *Fiabilité*
    - Tests unitaires
    - Tests d'intégration
  - *Maintenabilité*
    - Tests de non régression
- **Les métriques, les standards de codage, ...**
  - *Fiabilité*
  - *Maintenabilité*
- **& Le refactoring**
  - *Maintenabilité*

- **La méthodologie**
- **Normes**
  - **CMMi :** Modèle de référence, ensemble structuré de bonnes pratiques, destiné à appréhender, évaluer et améliorer les activités des entreprises d'ingénierie
  - **ISO/CEI 9126**

Mesurer la qualité logicielle

"You can't control what you can't measure." (Tom DeMarco)

FONDAZIONE BRUNO KESSLER

Qualité interne du code

- **Metriques**: couplage, cohesion ...
- **Code smells** et patterns
- **Conventions de codage**: règles de codage, de style, etc.

Mesurer la qualité : Métriques

Approche quantitative de la mesure de la qualité

http://www-igm.univ-mlv.fr/~dr/XPOSE2008/Mesure%20de%20la%20qualite%20du%20code%20source%20-%20Algorithmes%20et%20outils/files/mesure-qualite-code.pdf

Métrique Logicielle

- **Métrique : mesure d'une propriété d'un logiciel (par exemple le nombre de lignes de codes)**

- Approche quantitative : extraire une mesure de la qualité d'un logiciel à partir de l'analyse statistique du code source.
  - Avantage : simplicité de mise en oeuvre.
  - Principal problème : il faut trouver des **indicateurs significatifs** et les algorithmes correspondants.

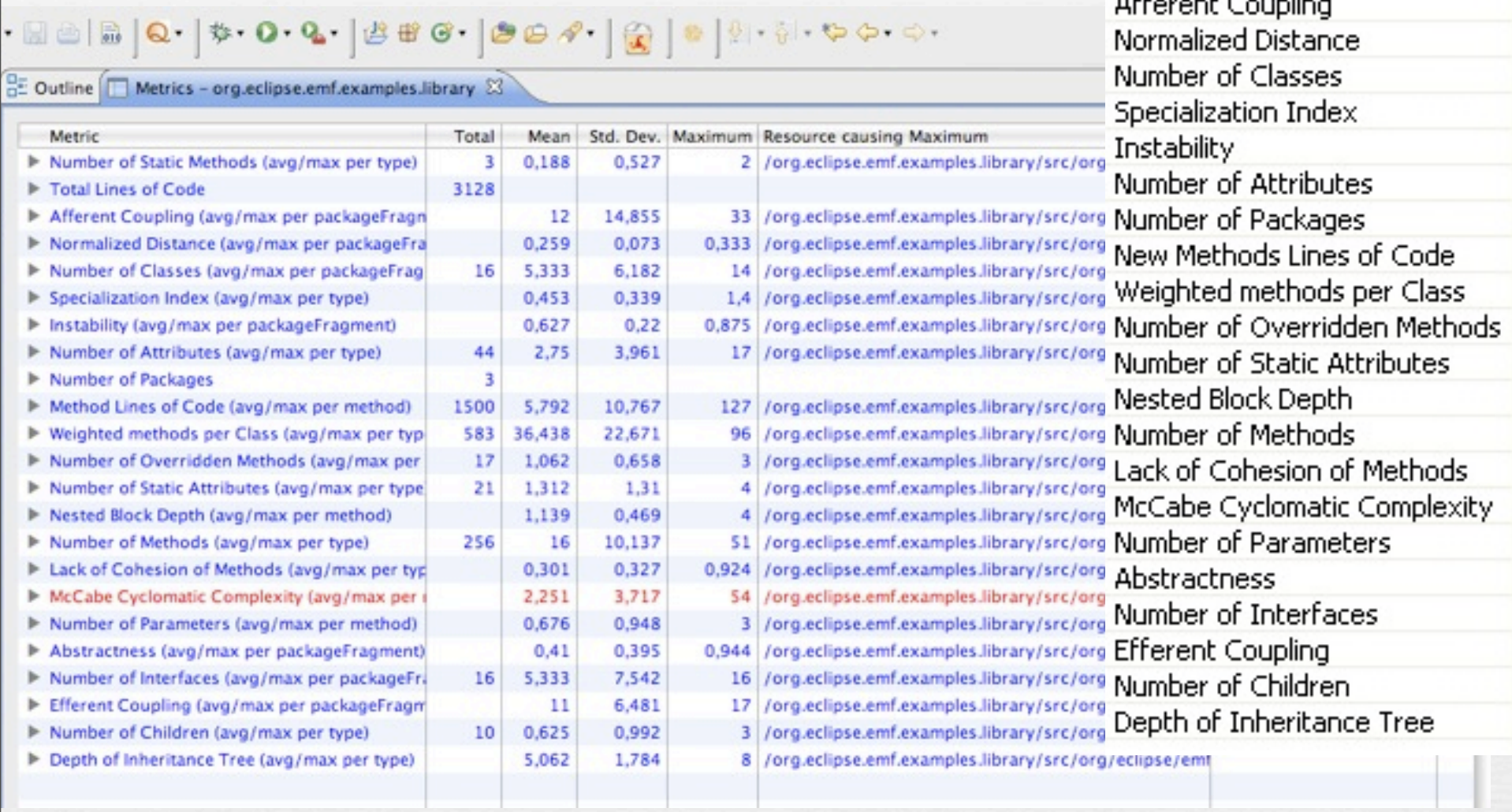
Exemples de métriques

- Nombre de Lignes de codes
- Nombre de méthodes par classe
- Couplage afférent/efférent
- Niveau d'abstraction
- Instabilité
- ....

➡Pas de métrique “absolue” : la pertinence de chaque métrique dépend du projet et surtout de l'interprétation qui en est faite.

Tests statiques

(plugin Metrics : http://metrics.sourcefor



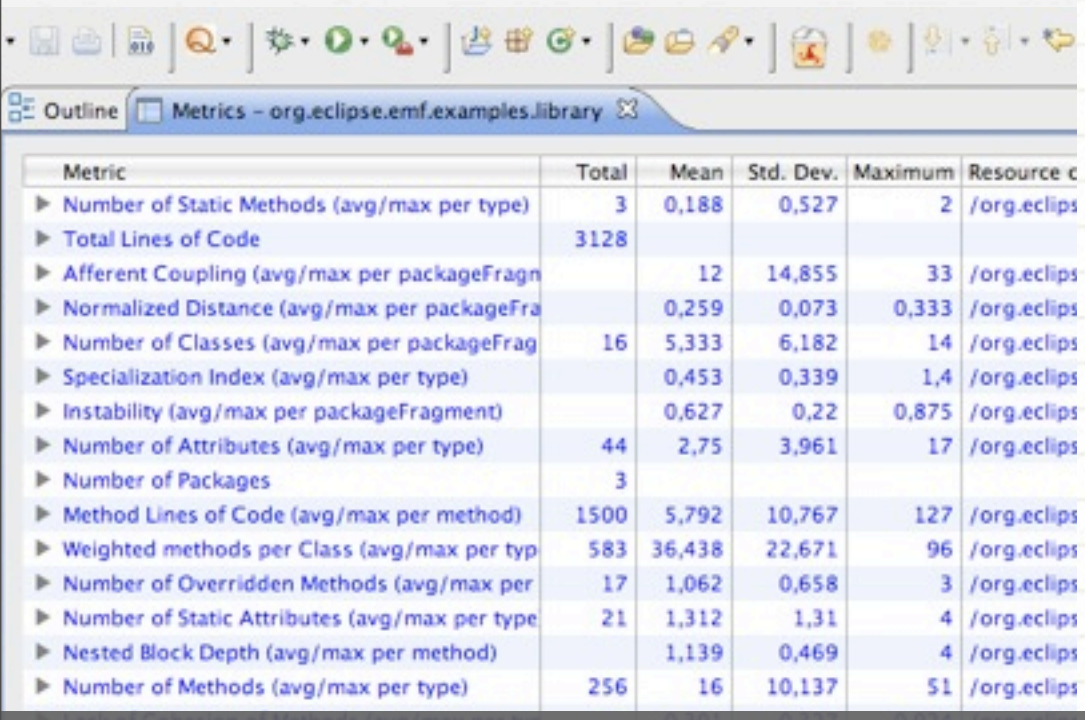
Outline | Metrics - org.eclipse.emf.examples.library

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum
Number of Static Methods (avg/max per type)	3	0,188	0,527	2	/org.eclipse.emf.examples.library/src/org
Total Lines of Code	3128				
Afferent Coupling (avg/max per packageFragn		12	14,855	33	/org.eclipse.emf.examples.library/src/org
Normalized Distance (avg/max per packageFra		0,259	0,073	0,333	/org.eclipse.emf.examples.library/src/org
Number of Classes (avg/max per packageFrag	16	5,333	6,182	14	/org.eclipse.emf.examples.library/src/org
Specialization Index (avg/max per type)		0,453	0,339	1,4	/org.eclipse.emf.examples.library/src/org
Instability (avg/max per packageFragment)		0,627	0,22	0,875	/org.eclipse.emf.examples.library/src/org
Number of Attributes (avg/max per type)	44	2,75	3,961	17	/org.eclipse.emf.examples.library/src/org
Number of Packages	3				
Method Lines of Code (avg/max per method)	1500	5,792	10,767	127	/org.eclipse.emf.examples.library/src/org
Weighted methods per Class (avg/max per typ	583	36,438	22,671	96	/org.eclipse.emf.examples.library/src/org
Number of Overridden Methods (avg/max per	17	1,062	0,658	3	/org.eclipse.emf.examples.library/src/org
Number of Static Attributes (avg/max per type	21	1,312	1,31	4	/org.eclipse.emf.examples.library/src/org
Nested Block Depth (avg/max per method)		1,139	0,469	4	/org.eclipse.emf.examples.library/src/org
Number of Methods (avg/max per type)	256	16	10,137	51	/org.eclipse.emf.examples.library/src/org
Lack of Cohesion of Methods (avg/max per typ		0,301	0,327	0,924	/org.eclipse.emf.examples.library/src/org
McCabe Cyclomatic Complexity (avg/max per		2,251	3,717	54	/org.eclipse.emf.examples.library/src/org
Number of Parameters (avg/max per method)		0,676	0,948	3	/org.eclipse.emf.examples.library/src/org
Abstractness (avg/max per packageFragment)		0,41	0,395	0,944	/org.eclipse.emf.examples.library/src/org
Number of Interfaces (avg/max per packageFr	16	5,333	7,542	16	/org.eclipse.emf.examples.library/src/org
Efferent Coupling (avg/max per packageFragn		11	6,481	17	/org.eclipse.emf.examples.library/src/org
Number of Children (avg/max per type)	10	0,625	0,992	3	/org.eclipse.emf.examples.library/src/org
Depth of Inheritance Tree (avg/max per type)		5,062	1,784	8	/org.eclipse.emf.examples.library/src/org/eclipse/emf

Number of Static Methods
Total Lines of Code
Afferent Coupling
Normalized Distance
Number of Classes
Specialization Index
Instability
Number of Attributes
Number of Packages
New Methods Lines of Code
Weighted methods per Class
Number of Overridden Methods
Number of Static Attributes
Nested Block Depth
Number of Methods
Lack of Cohesion of Methods
McCabe Cyclomatic Complexity
Number of Parameters
Abstractness
Number of Interfaces
Efferent Coupling
Number of Children
Depth of Inheritance Tree

Tests stati

(plugin Metrics : http://metr



Metric	Total	Mean	Std. Dev.	Maximum	Resource c
Number of Static Methods (avg/max per type)	3	0,188	0,527	2	/org.eclips
Total Lines of Code	3128				
Afferent Coupling (avg/max per packageFragn		12	14,855	33	/org.eclips
Normalized Distance (avg/max per packageFra		0,259	0,073	0,333	/org.eclips
Number of Classes (avg/max per packageFrag	16	5,333	6,182	14	/org.eclips
Specialization Index (avg/max per type)		0,453	0,339	1,4	/org.eclips
Instability (avg/max per packageFragment)		0,627	0,22	0,875	/org.eclips
Number of Attributes (avg/max per type)	44	2,75	3,961	17	/org.eclips
Number of Packages	3				
Method Lines of Code (avg/max per method)	1500	5,792	10,767	127	/org.eclips
Weighted methods per Class (avg/max per typ	583	36,438	22,671	96	/org.eclips
Number of Overridden Methods (avg/max per	17	1,062	0,658	3	/org.eclips
Number of Static Attributes (avg/max per type	21	1,312	1,31	4	/org.eclips
Nested Block Depth (avg/max per method)		1,139	0,469	4	/org.eclips
Number of Methods (avg/max per type)	256	16	10,137	51	/org.eclips

Number of Static Methods	type
Total Lines of Code	compilationUnit
Afferent Coupling	packageFragment
Normalized Distance	packageFragment
Number of Classes	compilationUnit
Specialization Index	type
Instability	packageFragment
Number of Attributes	type
Number of Packages	packageFragmentR
New Methods Lines of Code	method
Weighted methods per Class	type
Number of Overridden Methods	type
Number of Static Attributes	type
Nested Block Depth	method
Number of Methods	type
Lack of Cohesion of Methods	type
McCabe Cyclomatic Complexity	method
Number of Parameters	method
Abstractness	packageFragment
Number of Interfaces	compilationUnit

Number of Classes : Total number of classes in the selected scope

Number of Children : Total number of direct subclasses of a class. ...

Number of Interfaces : Total number of interfaces in the selected scope

Depth of Inheritance Tree (DIT) : Distance from class Object in the inheritance hierarchy.

Number of Overridden Methods (NORM) : Total number of methods in the selected ...

McCabe Cyclomatic Complexity : Counts the number of flows through a piece of code.

Weighted Methods per Class (WMC) : Sum of the McCabe Cyclomatic Complexity for all methods in a class

Indice de spécialisation

- Se calcule sur une classe entière (puis éventuellement moyenne pour le projet)
- Définition :

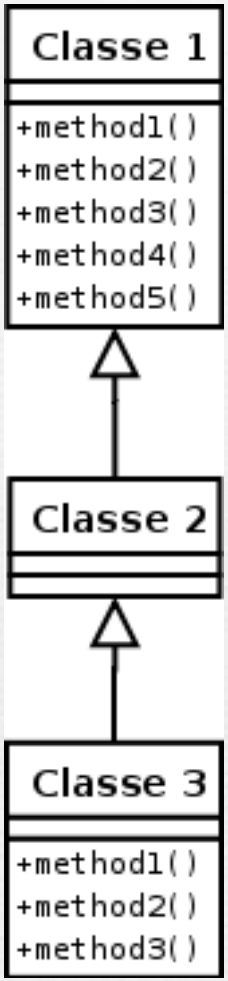
$$\frac{NORM \times DIT}{NOM}$$

- Avec
- NORM : Number of Overriden Methods
- DIT : Depth of Inheritance Tree (distance depuis la classe Object)
- NOM : Number of Methods

exemple de spécialisation

$$\frac{NORM \times DIT}{NOM}$$

- public class Identifiable implements Comparable<Identifiable>
  - 3 méthodes surchargées
  - Profondeur de l'arbre d'héritage : 1
  - 6 méthodes
  - 3/6 = 0,5



exemple de spécialisation

$$\frac{NORM \times DIT}{NOM}$$

- Classe 1 :

NORM : 0; DIT : 1; NOM : 5; specialisation : 0

- Classe 2 :

NORM : 0; DIT : 2; NOM : 0; specialisation : 0

- Classe 3 :

NORM : 3; DIT : 3; NOM : 3; specialisation : 3

exemple de spécialisation

$$\frac{NORM \times DIT}{NOM}$$

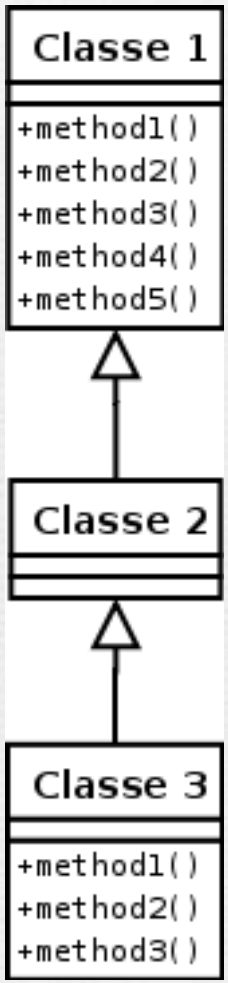
Augmente quand

- Le nombre de méthodes redéfinies ou la profondeur d'héritage augmente

Diminue quand

- Le nombre de méthodes spécifiques à la classe augmente.
- Le nombre de méthodes redéfinies diminue.

exemple de spécialisation



$$\frac{NORM \times DIT}{NOM}$$

Trop grand : la classe redéfinit trop de méthodes dont elle hérite : il faut penser à refactoriser en utilisant des interfaces par exemple.

- Moyenne : 0.05

Indice d'instabilité de packages

- Se calcule sur un paquetage ou un ensemble de paquetages.

$$\frac{Ce}{Ca+Ce}$$

- Définition :
  - Ce (efferent coupling) : le nombre de classes de ce paquetage qui dépendent de classes de l'extérieur. (indépendance)
  - Ca (afferent coupling) : le nombre de classes de l'extérieur qui dépendent de classes dans ce paquetage (responsabilité)

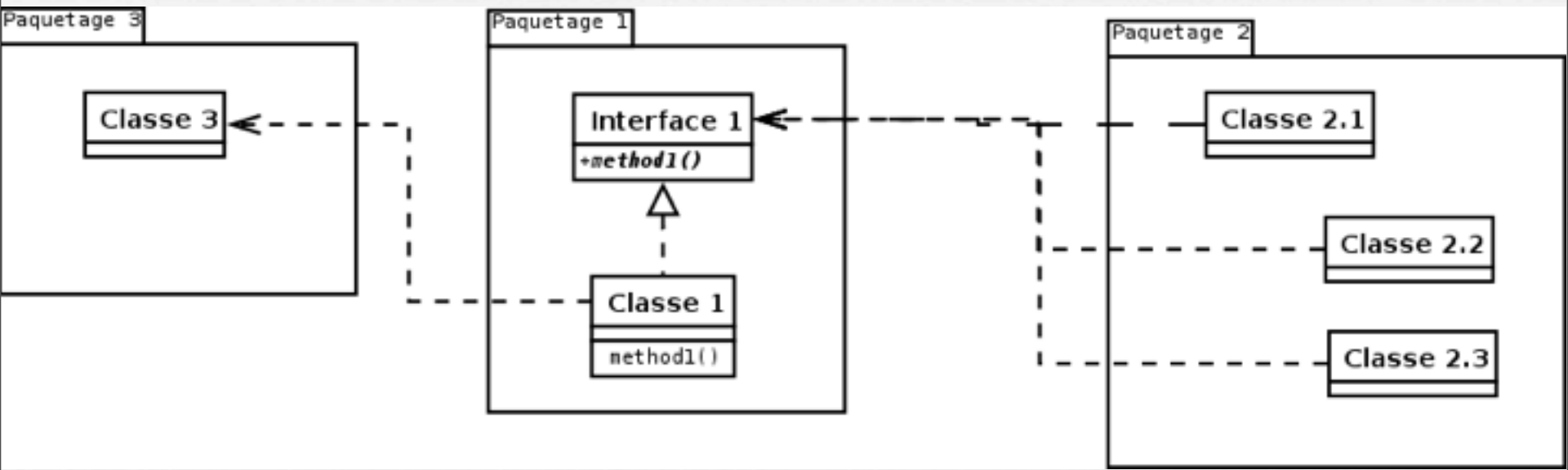
Indice d'instabilité d'un package

Par exemple, le package ReseauRoutier dépend de
- Ce = 4 classes extérieures (Arc, Sommet, Graphe, Parcours)
- Ca = 0, il n'est pas utilisé de l'extérieur.

Instabilité = 1

$$\frac{Ce}{Ca + Ce}$$

Indice d'instabilité d'un package



Package 3 : Ce = 0; Ca =1; Instabilité = 0
Package 1 : Ce = 1; Ca =3; Instabilité = 1/4 =0,25
Package 2 : Ce = 3; Ca =0; Instabilité = 1
Packages : Instabilité = 0.41

Indice d'instabilité d'un package

$$\frac{Ce}{Ca + Ce}$$

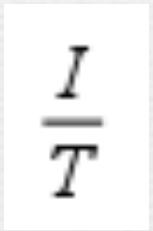
Indice d'instabilité - interprétation

- Indice compris entre 0 et 1 :
  – 0 : le paquetage est stable.
  – 1 : le paquetage n'est pas stable.

- Pour qu'un paquetage soit considéré comme stable avec cet indice, il faut qu'il y ait plus de dépendances entrantes que sortantes

Niveau d'abstraction de packages

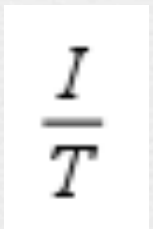
- Se calcule sur un paquetage ou un ensemble de paquetages.



- Avec
- – I : Le nombre d'interfaces et de classes abstraites.
- – T : le nombre total de types.

Niveau d'abstraction de packages

➡Se calcule sur un paquetage ou un ensemble de paquetages.



• S'utilise surtout comme élément de mesure de la “normalized distance from the main sequence”

Distance from the main sequence

- Se calcule sur un paquetage ou un ensemble de paquetages.
- Définition :

$$|Abstractness + Instability - 1|$$

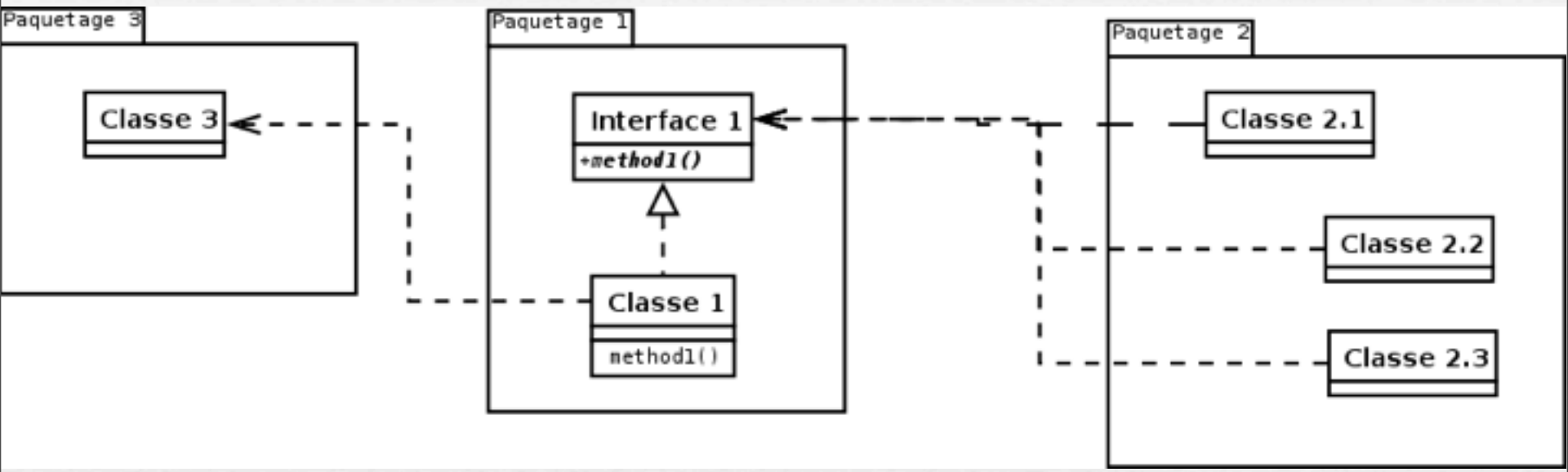
- Avec
- – Abstractness : l'indice d'abstraction du paquetage.
- – Instability : l'indice d'instabilité du paquetage.

Distance from the main sequence

$$|Abstractness + Instability - 1|$$

➡ Un paquetage est bien conçu si ce nombre est proche de zéro.

Distance from the main sequence



Package 3 : asbtraction=0; instabilité=0; distance = 1
Package 1 : asbtraction=0,5; instabilité=0,25; distance = 0,25
Package 2 : asbtraction = 0; instabilité = 1; distance = 0
Packages : distance = 0,417

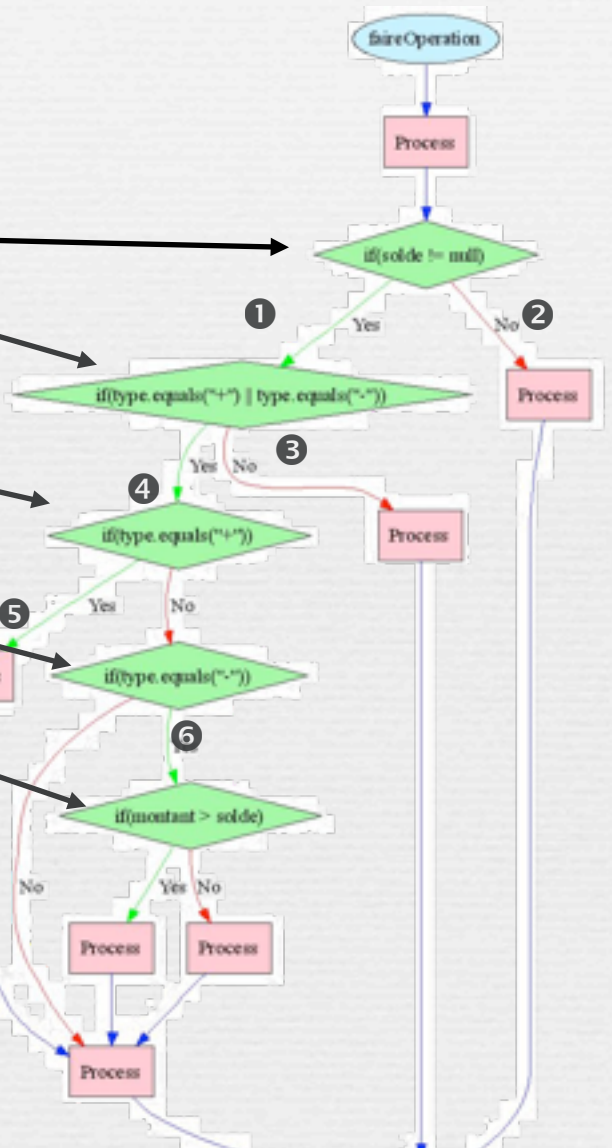
$$|Abstractness + Instability - 1|$$

Assurer une bonne lisibilité du code

- **Complexité cyclomatique d'une méthode**
  - Définition
    - Nombre de chemins linéairement indépendants qu'il est possible d'emprunter dans cette méthode
      - Nombre de points de décision de la méthode : if, case, while,… + 1 (le chemin principal)
  - Interprétation
    - Une méthode avec une haute complexité cyclomatique est plus difficile à comprendre et à maintenir
      - > à 30 : refactoriser la méthode
      - < à 30 : acceptable si suffisamment de tests
    - La complexité cyclomatique est liée à la notion de « couverture de code"
      - Nombre de tests unitaires = sa complexité cyclomatique
        - **Un chemin = un test**

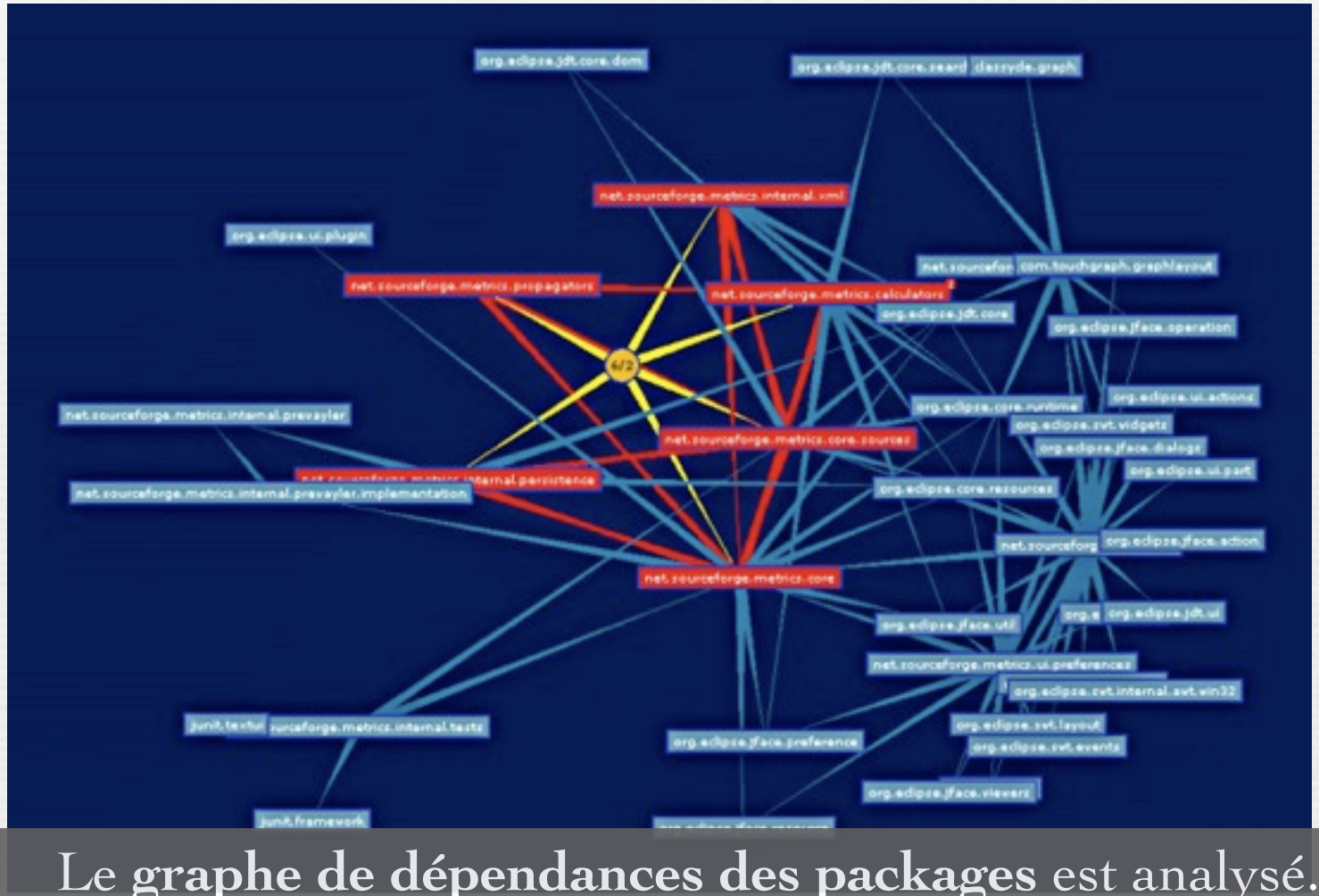
Complexité cyclomatique d'une méthode

```
package banque;


public class Banque {
   private Double solde;

   public void faireOperation(
        String type, double montant) {
      System.out.println("Début d'opération.");
     if(solde != null) {
      if(type.equals("+") || type.equals("-"))
      {
       if(type.equals("+")) {
         solde += montant;
       }
       if(type.equals("-")) {
         if(montant > solde) {
           System.err.println("!!!");
         }
         else {
           solde -= montant;
         }
       }
      }
      else {
       System.err.println("...");
      }
     }
     else {
      System.err.println(".");
     }
    System.out.println("Fin d'opération.");
   }
}
```



Analyses des dépendances de packages

(plugin Metrics : http://metrics.sourceforge.net/)



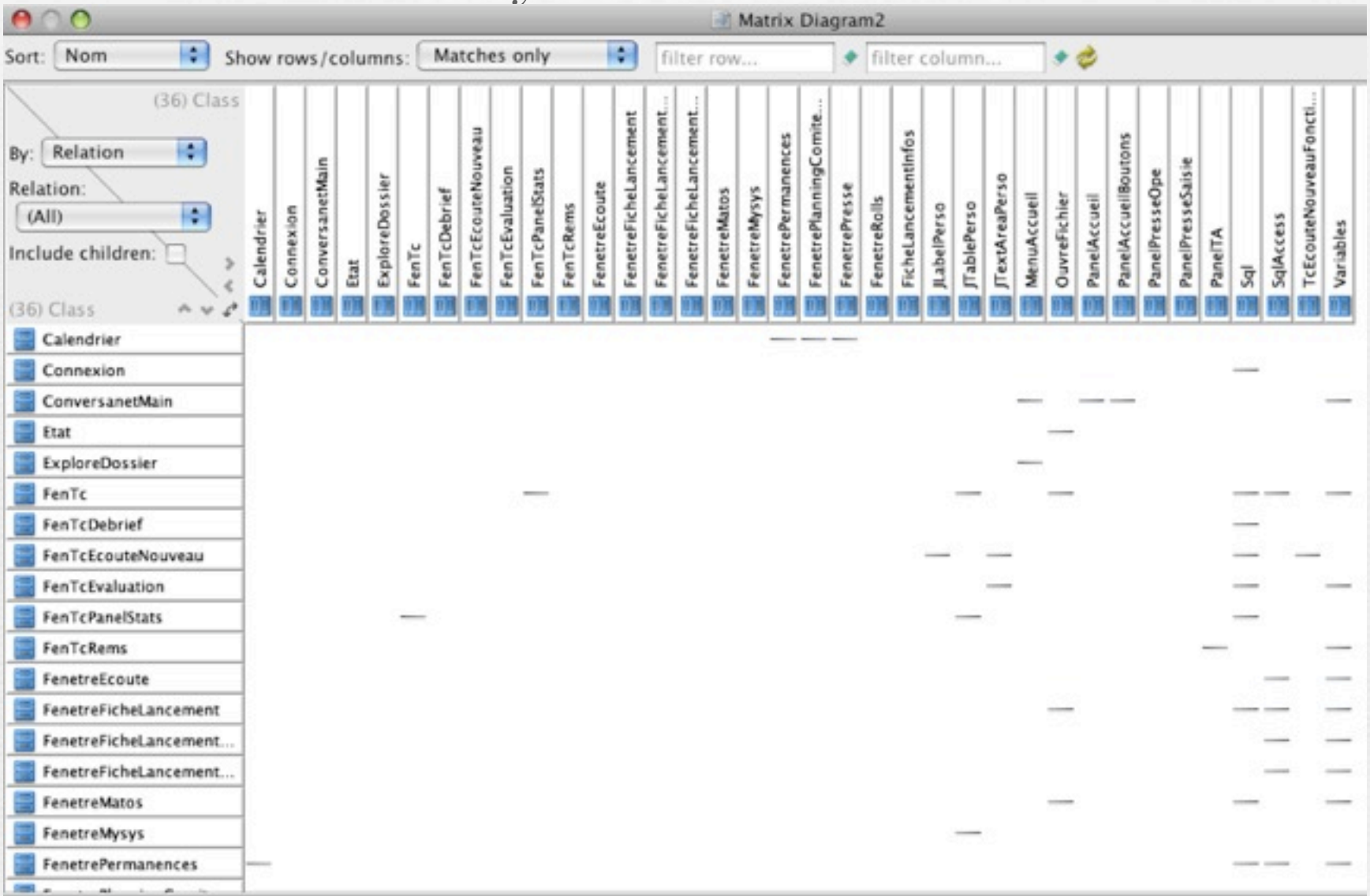
Le **graphe de dépendances des packages** est analysé.
Les packages fortement connectés sont colorés en rouge. Les autres packages ne participant pas à des cycles sont de couleur bleue.

Metrics compress the system into numbers

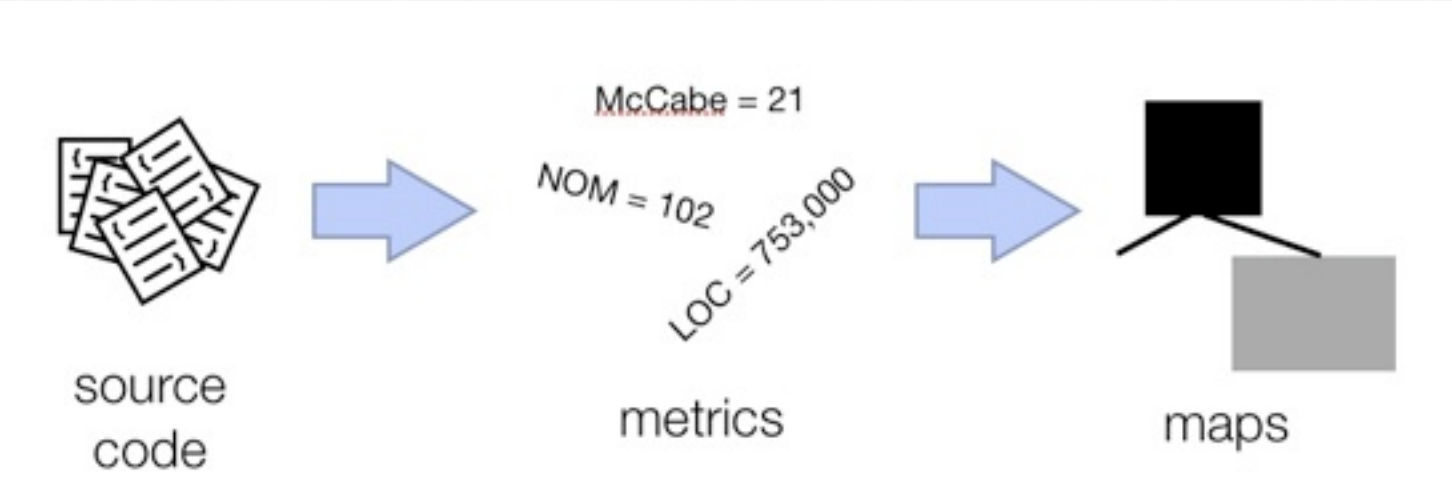
NOM	NOC	DUPLINES
LOC	NOCmts	NAI
TCC	NOPA	NOA
WMC	WLOC	NI
CYCLO	WNOC	...
ATFD	WOC	
HNL	MSG	

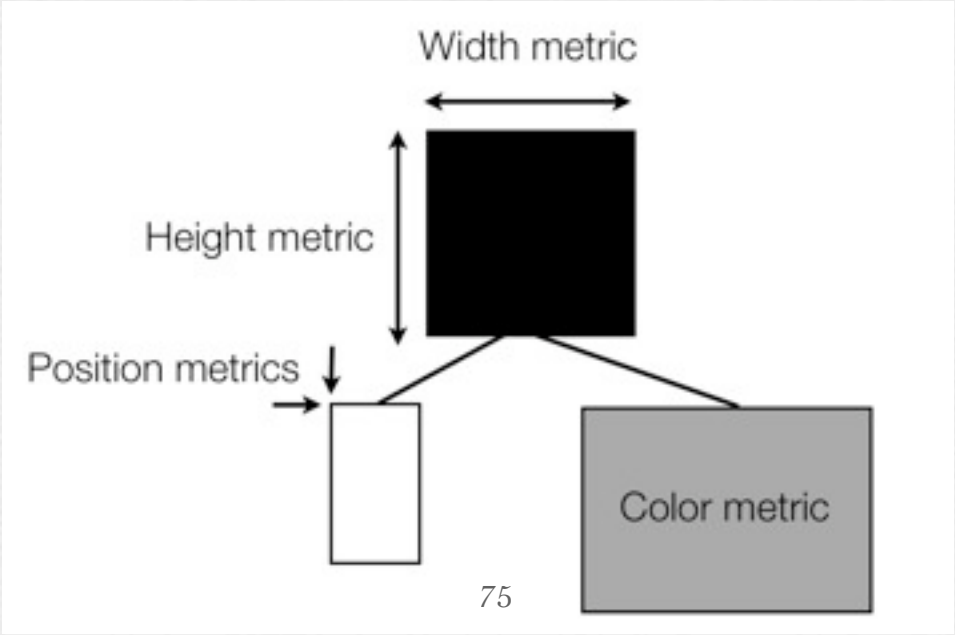
Tests statiques

Analyse de modèles avec VP

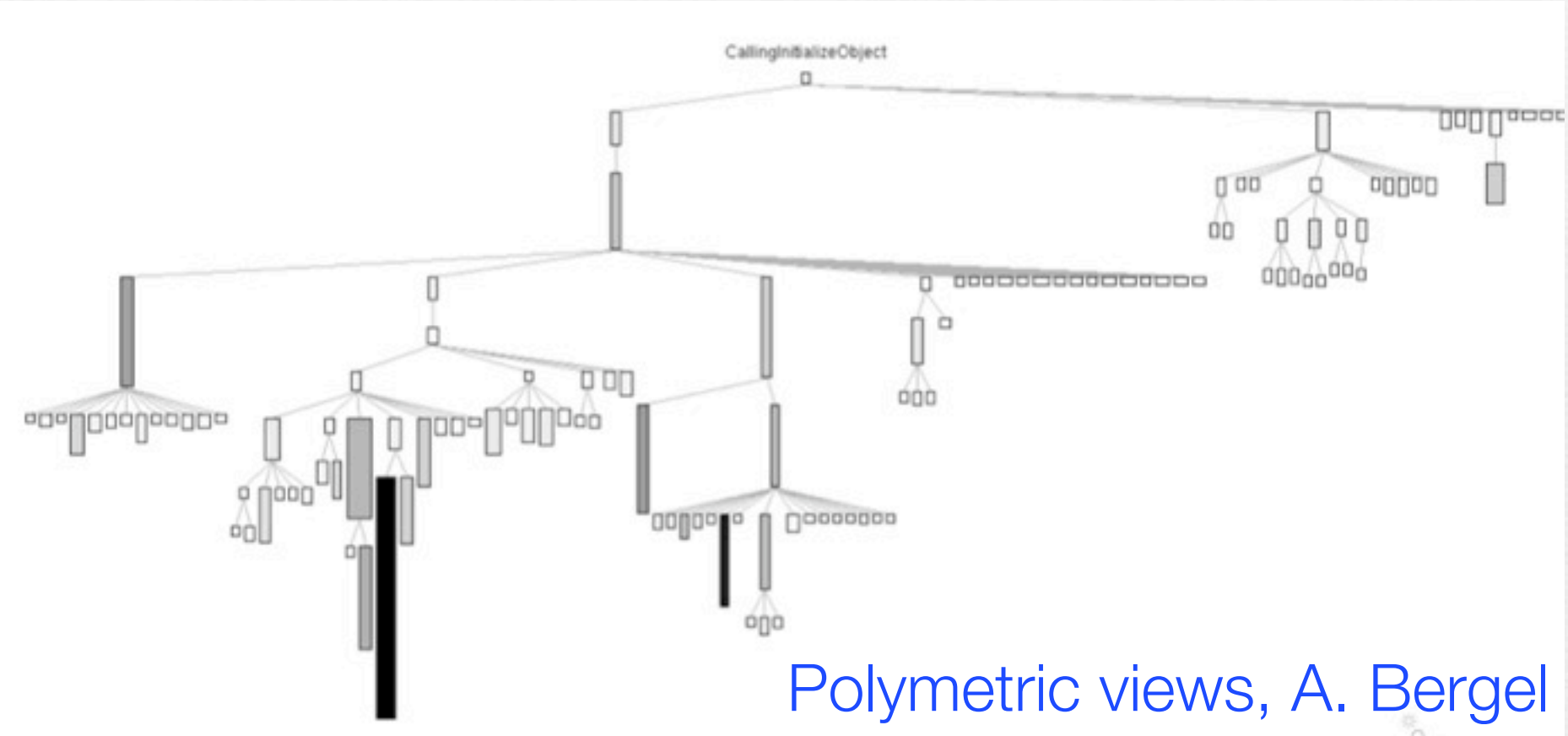


Tests statiques
Autres Visualisations



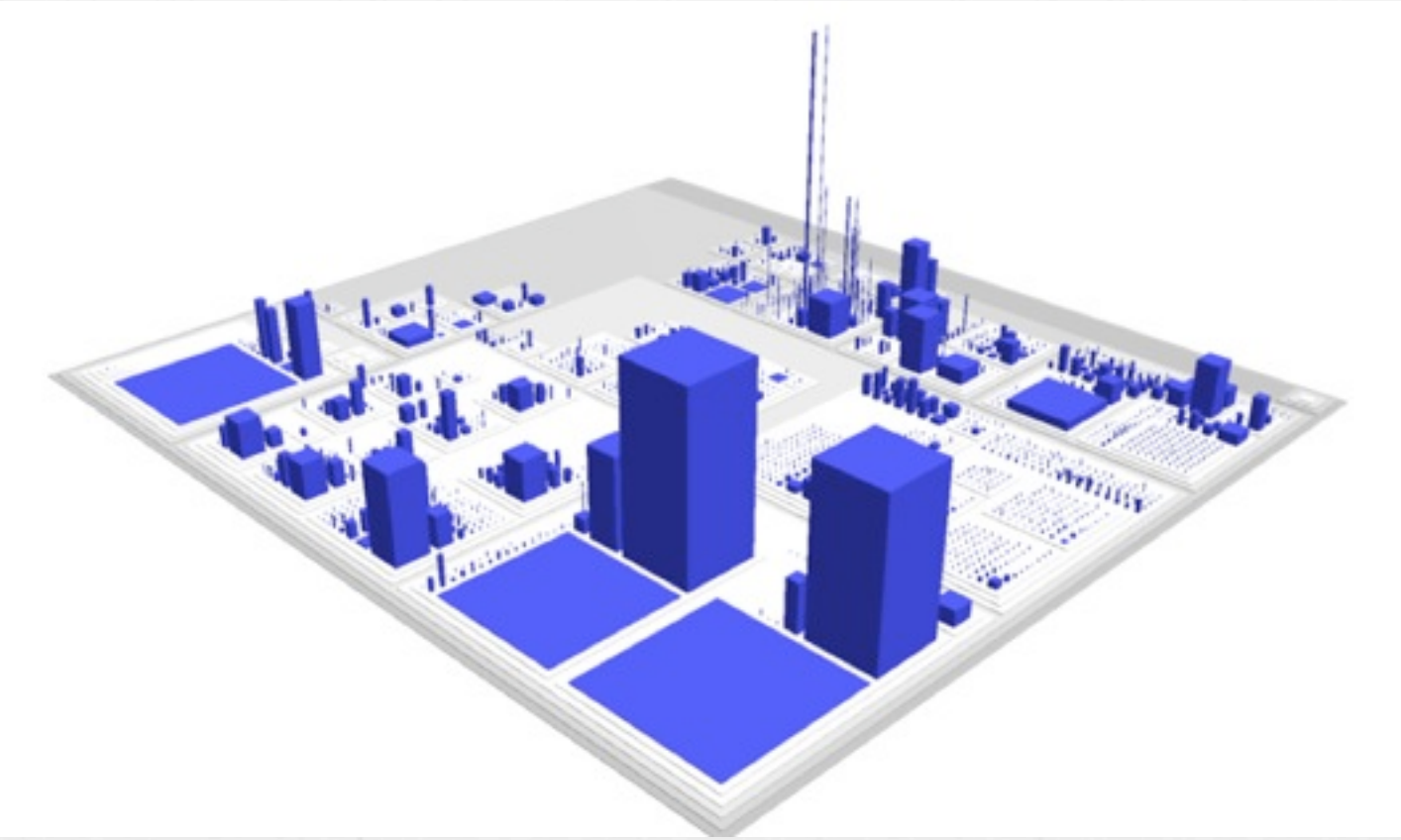


Tests statiques
Autres Visualisations



Polymetric views, A. Bergel

Tests statiques
Autres Visualisations



CodeCity (Wettel, Univ Lugano)

http://www.inf.usi.ch/phd/wettel/codecity-wof.html

Analyse du style du code

Assurer une bonne lisibilité du code

- S'accorder sur un ensemble de pratiques permettant d'assurer la bonne compréhension du code
- Utiliser des standards de
  - Codage : Entreprise, Projet
  - Nommage
  - Indentation

Est-ce la classe que vous cherchez?

```
public class T01<T02,T03> extends T04<T02,T03>
  implements T05<T02,T03>, T06, T07 {


  public T03 m01(T02 x01, T03 x02) {
    if (x01 == null)
      return m02(x02);
    int x03 = m03(x01.m04());
    int x04 = m05(x03, x05.x06);
    for (T08<T02,T03> x07 = x08[x04]; x07 != null; x07 = x07.x09) {
      T09 x10;
      if (x07.x11 == x03 && ((x10 = x07.x12) == x01 || x01.m06(x10))) {
        T03 x13 = x07.x14;
        x07.x14 = x02;
        x07.m07(this);
        return x13;
      }
    }
    x15++;
    m08(x03, x01, x02, x04);
    return null;
  }
}
```

Is this one the class you are looking for?

```
public class HashMap<K,V> extends AbstractMap<K,V>
  implements Map<K,V>, Cloneable, Serializable {


  public V put(K key, V value) {
    if (key == null)
      return putForNullKey(value);
    int hash = hash(key.hashCode());
    int i = indexFor(hash, table.length);
    for (Entry<K,V> e = table[i]; e != null; e = e.next) {
      Object k;
      if (e.hash == hash && ((k = e.key) == key || key.equals(k))) {
        V oldValue = e.value;
        e.value = value;
        e.recordAccess(this);
        return oldValue;
      }
    }
    modCount++;
    addEntry(hash, key, value, i);
    return null;
  }
}
```

13

Identifiant auto-documenté

Bons identifiants:
- fournit des indices concis sur la sémantique des données identifiées;
- évite aux programmeurs de lire le segment de code en entier;
- accélère l'acquisition de connaissances;
- facilite la compréhension du programme (recherche de mots, ...)

Checkstyle

- http://checkstyle.sourceforge.net/

- Checkstyle performs source code analysis.
  - ✓ Originally for "coding standard" (formatting)
  - ✓ Now includes design-level best practice compliances.
- Classes of checks:
  - ✓ JavaDoc, Naming Conventions, Headers, Size Violations, Imports, WhiteSpace, Modifers, Block Checks, Coding, Class Design, Duplicate Code, Metrics, J2EE.
- Can be extended with new checks.
  - ✓ Use a configuration file to customize what checks your system should comply with.

http://www.oracle.com/technetwork/articles/java/index-137868.html

Java conventions

CITS1220 Software Engineering Coding Style Rules

		Rules	Example
Naming Conventions	Classes	• Use nouns • Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML). ✓ Begin with upper case letters • First letter of each internal word capitalized	QueueBlock, ReversalADT, LinkedList ,CustomerAccount, not MaintainCustomerData
	Methods	• Use verbs ✓ Begin with lower case letters • First letter of each internal word capitalized	examine, delete, isEmpty, toString
	Instance Variables	• Use nouns, plurals for collection classes ✓ Begin with lower case letters • First letter of each internal word capitalized • Dos not start with underscore _ or dollar sign $ characters, even though both are allowed. • One-character variable names should be avoided except for common temporary variables.	items, firstAction, list Common temporary variables names **Integers**: i, j, k, m, and n **Characters** : c, d, and e
	Constants	✓ All characters in upper case • Words are separated by underscores (" ")	`private static final int MAX_SEQUENCE`
	Files	✓ File names should be the same as the (principal) class stored in the file.	

Layout	Classes & Interfaces	✓ No space between a method name and the parenthesis "(" starting its parameter list
		✓ Open brace "{" appears at the end of the same line as the declaration statement
		✓ Closing brace "}" starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the "}" should appear immediately after the "{"
		✓ The internal statements are indented by 2 spaces.
			`public class QueueBlock {`
			`.`
			`.`
			`protected Object [] items;`
			`.`
			`.`
			
			`public int isZero (){return 0;}`
			`.`
			`.`

Java doc

Order of Tags
Include tags in the following order:

1. `@author` (classes and interfaces only, required)
2. `@version` (classes and interfaces only, required. See footnote 1)
3. `@param` (methods and constructors only)
4. `@return` (methods only)
5. `@exception` (@throws is a synonym added in Javadoc 1.2)
6. `@see`
7. `@since`
8. `@serial` (or @serialField or @serialData)
9. `@deprecated` (see How and When To Deprecate APIs)

@version (reference page)
The Java Software convention for the argument to the @version tag is the SCCS string "%I%, %G%", which converts to something like "`1.39, 02/28/97`" (mm/dd/yy) when the file is checked out of SCCS.

```
* All coordinates which appear as arguments to the methods of this
 * Graphics object are considered relative to the translation origin
 * of this Graphics object prior to the invocation of the method.
 * All rendering operations modify only pixels which lie within the
 * area bounded by both the current clip of the graphics context
 * and the extents of the Component used to create the Graphics object.
 *
 * @author      Sami Shaio
 * @author      Arthur van Hoff
 * @version     %I%, %G%
 * @since       1.0
 */
```

http://www.oracle.com/technetwork/articles/java/index-137868.html

Java doc

Tag Comments
As a reminder, the fundamental use of these tags is described on the Javadoc Reference page. Java Software generally uses the following additional guidelines to create comments for each tag:

@author (reference page)
You can provide one @author tag, multiple @author tags, or no @author tags. In these days of the community process when development of new APIs is an open, joint effort, the JSR can be consider the author for new packages at the package level. For example, the new package java.nio has "@author JSR-51 Expert Group" at the package level. Then individual programmers can be assigned to @author at the class level. As this tag can only be applied at the overview, package and class level, the tag applies only to those who make significant contributions to the design or implementation, and so would not ordinarily include technical writers.

The @author tag is not critical, because it is not included when generating the API specification, and so it is seen only by those viewing the source code. (Version history can also be used for determining contributors for internal purposes.)

If someone felt strongly they need to add @author at the member level, they could do so by running javadoc using the new 1.4 `-tag` option:

```
-tag author:a:"Author:"
```

If the author is unknown, use "unascribed" as the argument to @author. Also see order of multiple @author tags.

@version (reference page)
The Java Software convention for the argument to the @version tag is the SCCS string "%I%, %G%", which converts to something like "`1.39, 02/28/97`" (mm/dd/yy) when the file is checked out of SCCS.

http://www.oracle.com/technetwork/articles/java/index-137868.html

PMD

PMD

- PMD also performs source code analysis.
  - More 'design' oriented than Checkstyle.
  - Lots of overlap.
- PMD rulesets:
  - Basic, Braces, Code Size, Clone, Controversial, Coupling, Design, Finalizers, Import, J2EE, JavaBeans, JUnit, Logging, Migrating, Naming, Optimizations, Exceptions, Strings, Security, Unused Code, JSP, JSF.

On retient au moins

- Quelles sont les propriétés que je peux/dois vérifier lorsque je fournis un logiciel?
- Comment vérifier que mon code vérifie bien ces propriétés?
  - ✓ Ne pas se poser ces questions, revient à faire un gâteau sans se poser les questions :
    - Pour qui est ce gâteau? Pour combien de personnes? Pour quelle occasion ? Doit-il être transporté?
    - Puis/dois je y mettre du chocolat?
    - Est-il assez/trop sucré, y-en a-t-il assez pour tous, va-t-il résister au transport?