Mesure de la qualité du code source

ALGORITHMES ET OUTILS

Introduction

- La qualité logicielle dépend de nombreux facteurs :
  - Qualité de projet :
    Portabilité, fiabilité, la tolérance de pannes, la simplicité, la facilité de correction, sécurité...
  - Qualité du code source :
    - Clarté/concision
    - Complexité
    - Couverture par les tests

Métriques Approche quantitative de la mesure de la qualité

Approche quantitative

- Métrique logicielle : 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.

Métriques

- Exemples de métriques :
  - Lignes de codes
  - Nombre de méthodes par classe
  - 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.

Métriques courantes

Indice de spécialisation

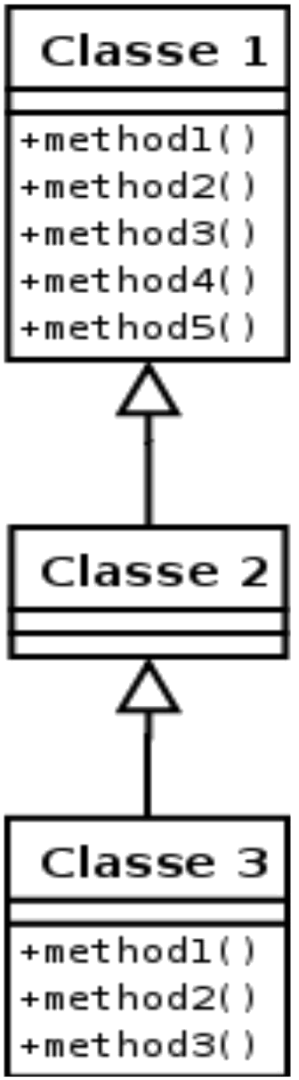
- Se calcule sur une classe entière (puis éventuellement moyenne pour le projet)
- Définition :

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

Avec

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

Indice de spécialisation - calcul



Indice de spécialisation

- 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.

Indice de spécialisation - Interprétation

- 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

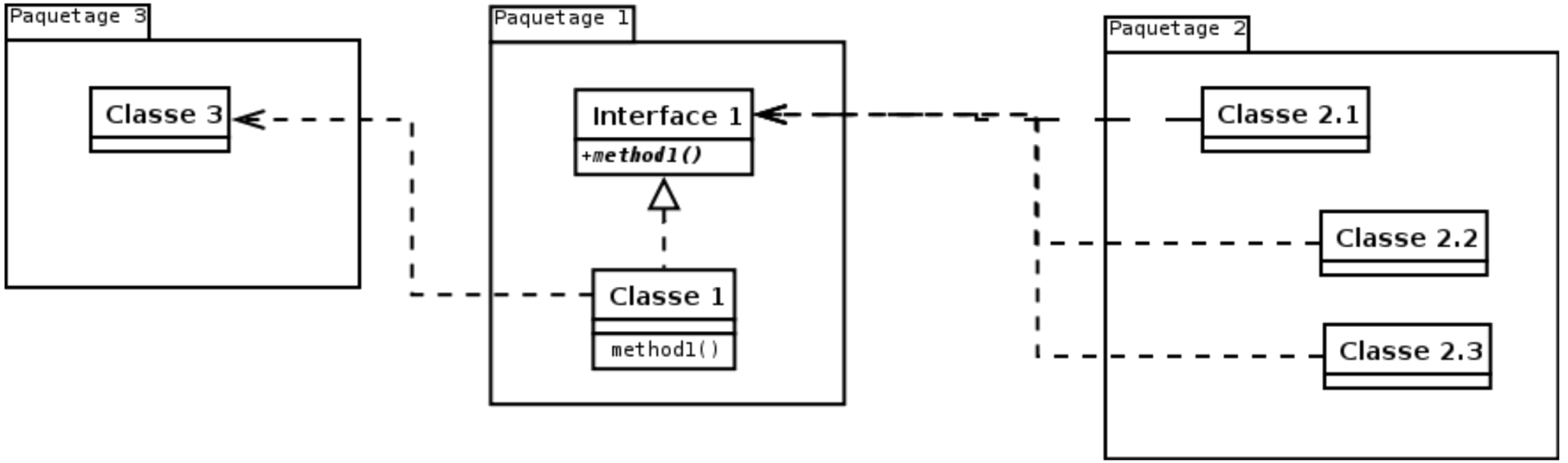
Indice d'instabilité

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

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

  - Ca : Couplage afferent, le nombre de classes en dehors du paquetage qui dépendent de classes de ce paquetage
  - Ce : Couplage efferent, le nombre de classes de ce paquetage qui dépendent de classes en dehors de ce paquetage.

Indice d'instabilité - calcul



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

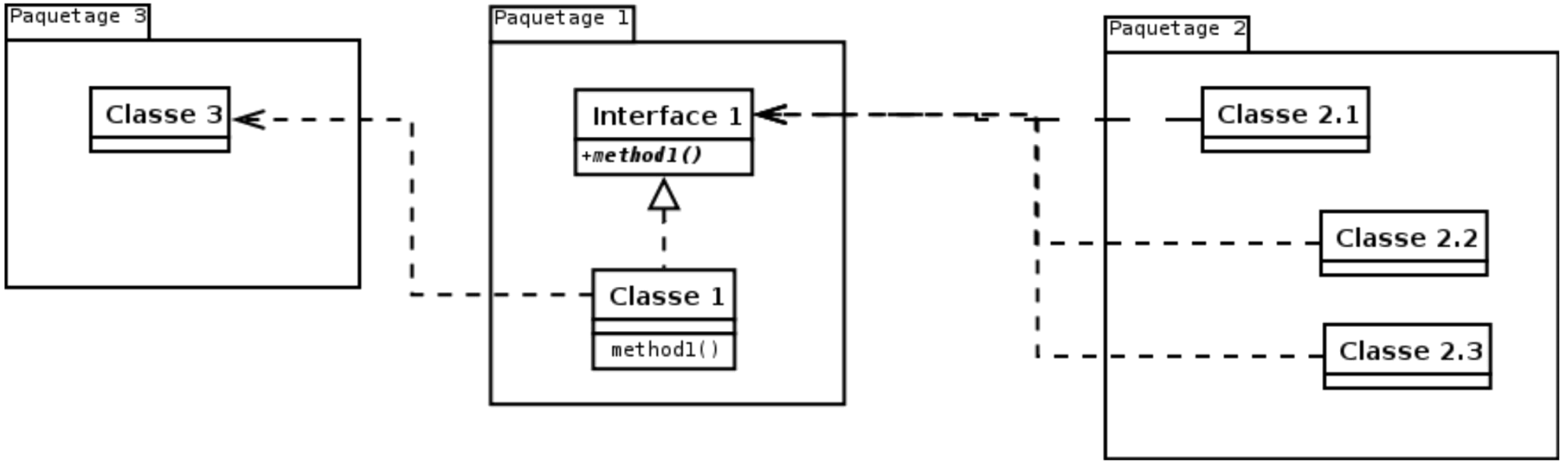
- Se calcule sur un paquetage ou un ensemble de paquetages.
- Définition :

$$\frac{I}{T}$$

Avec

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

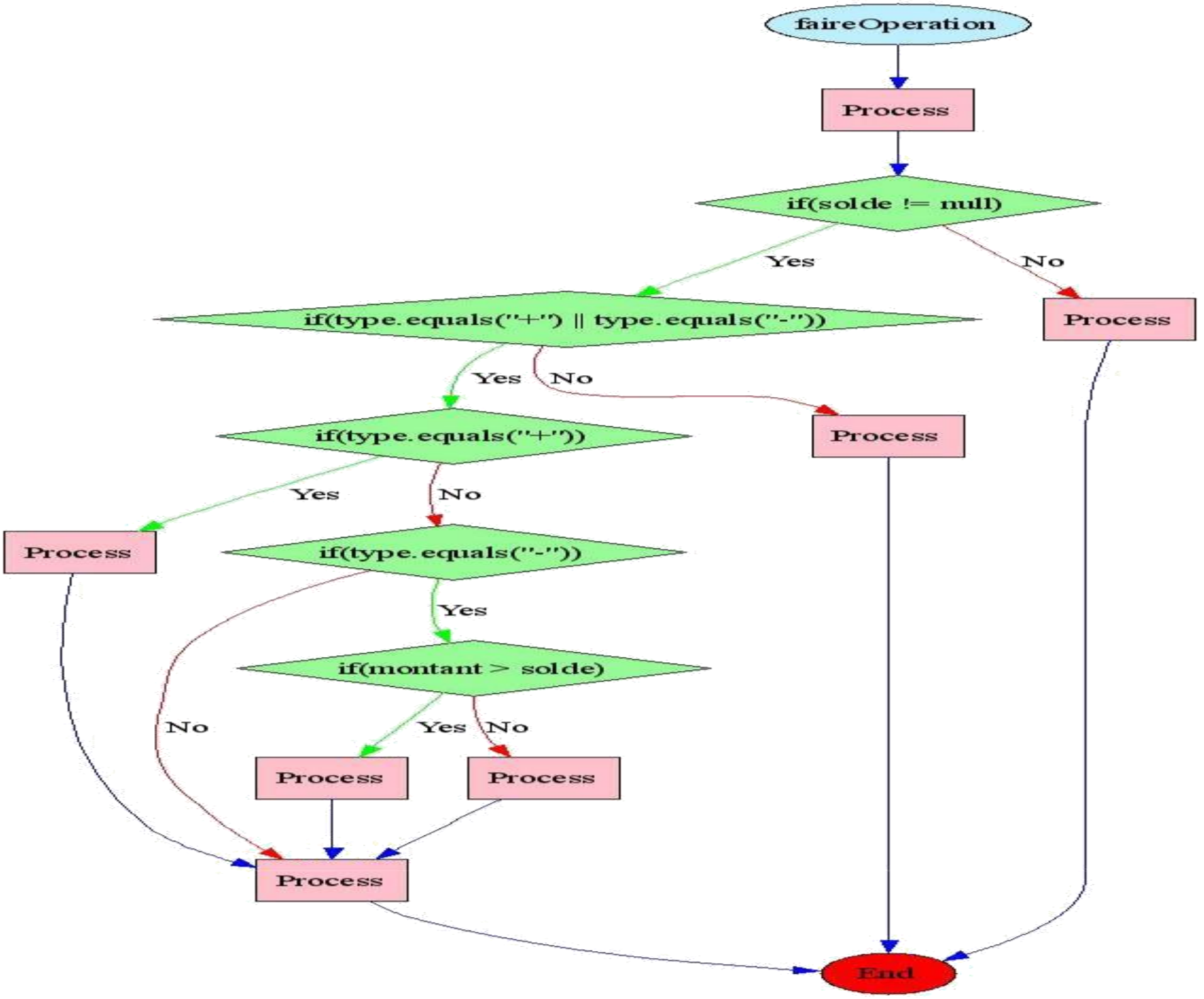
Niveau d'abstraction - calcul



Compléxité cyclomatique

- Se calcule sur une méthode
- Indice très courant dans les programmes d'analyse de code source
- C'est le nombre de chemins linéairement indépendants qu'il est possible de suivre au sein d'une méthode

Calcul d'une complexité cyclomatique


faireOperation
Process
if(solde != null)
Yes
No
if(type.equals("+") || type.equals("-"))
Process
Yes
No
if(type.equals("+"))
Process
Yes
No
Process
if(type.equals("-"))
Yes
if(montant > solde)
No
Yes
No
Process
Process
Process
End

Compléxité cyclomatique - Algorithme

- Calcul simple de la complexité cyclomatique :
  - Il suffit de compter le nombre de points de décision (if, case, while, ...)

Compléxité cyclomatique - Interprétation

- Un module avec une haute complexité cyclomatique est plus difficile à comprendre.
- Si une méthode a une complexité cyclomatique trop élevée (au delà de 30), elle doit être refactorisée.

- Une complexité cyclomatique inférieure à 30 est acceptable si la méthode est suffisament testée.

Compléxité cyclomatique - Interprétation

- Notion de “Code Coverage” :
  - Pourcentage de chemins couvert par les tests.
  - A 100%, le nombre de tests unitaires d'une méthode est égal à son indice de complexité cyclomatique.

Autres indicateurs “triviaux”

- Ratio lignes de commentaires/nombre de lignes
- Pourcentage de méthodes trop longues
- Nombre de classes par paquetage
- Nombre de méthodes par classe
- ...

Outils

Outils

- En Java, un grand nombre d'outils libres sont disponibles.
  - Cobertura
  - Crap4J
  - PMD
  - FindBugs
  - Eclipse Metrics
  - SonarQube
  - ....

Crap4J

- Utilise une formule spécifique :

$$CRAP(m) = comp(m)^2 \cdot \left(1 - \frac{cov(m)}{100}\right)^3 + comp(m)$$

- Une méthode doit être refactorisé si elle possède un indice de “crappiness” au dessus de 30.

Crap4J

- Crap4J autorise les complexités cyclomatiques élevées si le code est bien couvert par les tests.

Complexité Cyclomatique	**Pourcentage de couverture par les tests requis**
0 – 5	0%
6 - 10	42%
11-15	57%
16-20	71%
21-25	80%
26-30	100%
31+	-

PMD

- Détection d'un certain nombre d'anti patterns, connus pour poser problème.
- L'ensemble des règles que PMD vérifie est disponible à l'adresse suivante :

  http://pmd.sourceforge.net/rules/index.html

- Vérifie énormément de choses :
  - EmptyCatchBlock
  - UnnecessaryParentheses
  - CallSuperInConstructor

FindBugs

- Semblable à PMD dans le principe : détection d'anti patterns.
- Liste des bugs détectés disponible sur le web

  http://findbugs.sourceforge.net/bugDescriptions.html
- Exemple d'anti-pattern détecté :
  - Null pointer dereference
  - Method does not check for null argument
  - Read of unwritten field

Eclipse Metrics

- Permet de calculer beaucoup de metrics :
  - Compléxité cyclomatique
  - Nombre de ligne de codes
  - Indice de spécialisation
  - Indice d'instabilité
  - Niveau d'abstraction
  - ...

Démo

Outil SonarQube