Anti-patrons
= défauts de conception

Naouel Moha

Université du Québec à Montréal

Polytech Nice Sophia, Mardi 13 décembre 2016

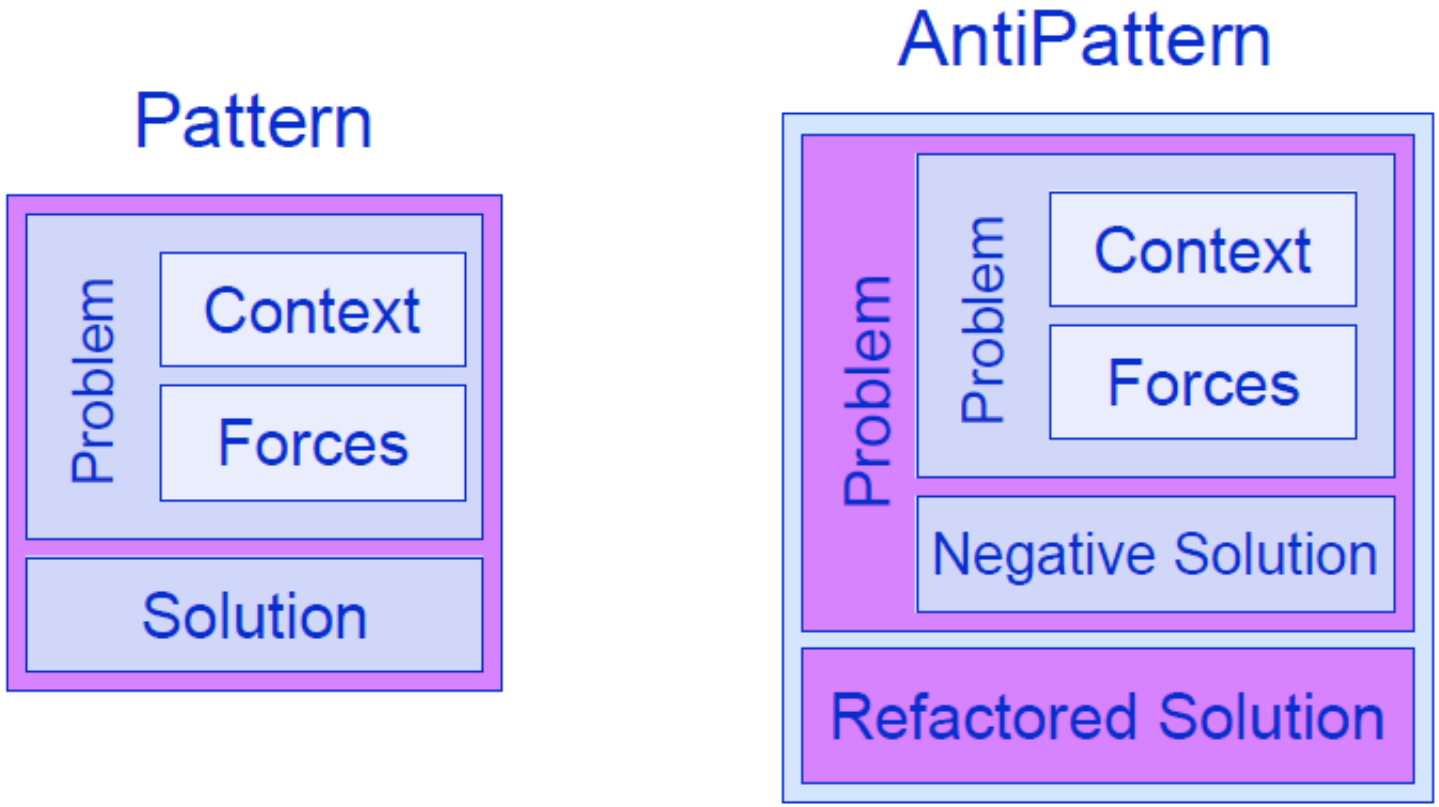
UQÀM LATECE

Défauts de conception

- Patrons de conception sont de “bonnes” solutions à des problèmes de conception récurrents
- Défauts de conception
  - sont de “mauvaises” solutions, mauvaises pratiques de conception qui mènent à des conséquences négatives
  - 2 catégories:
    - Défauts de haut niveau : anti-patrons
    - Défauts de bas niveau : mauvaises odeurs

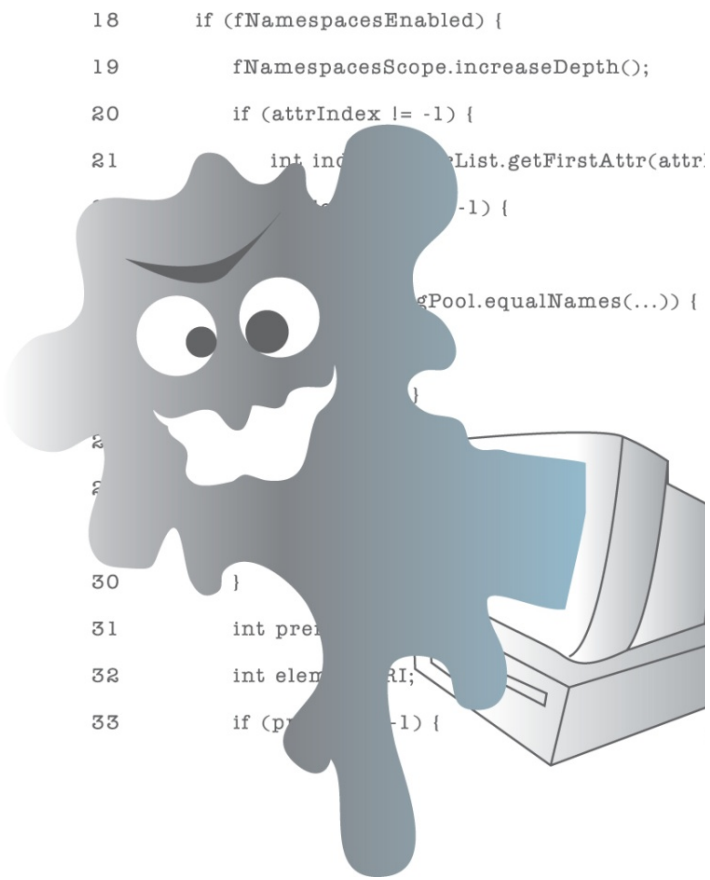
Défauts de conception

- Un anti-patron est un type spécial de patron de conception caractérisé par une solution refactorisée



Défauts de conception

- 2 exemples d'anti-patrons
  - Blob *(God Class)*



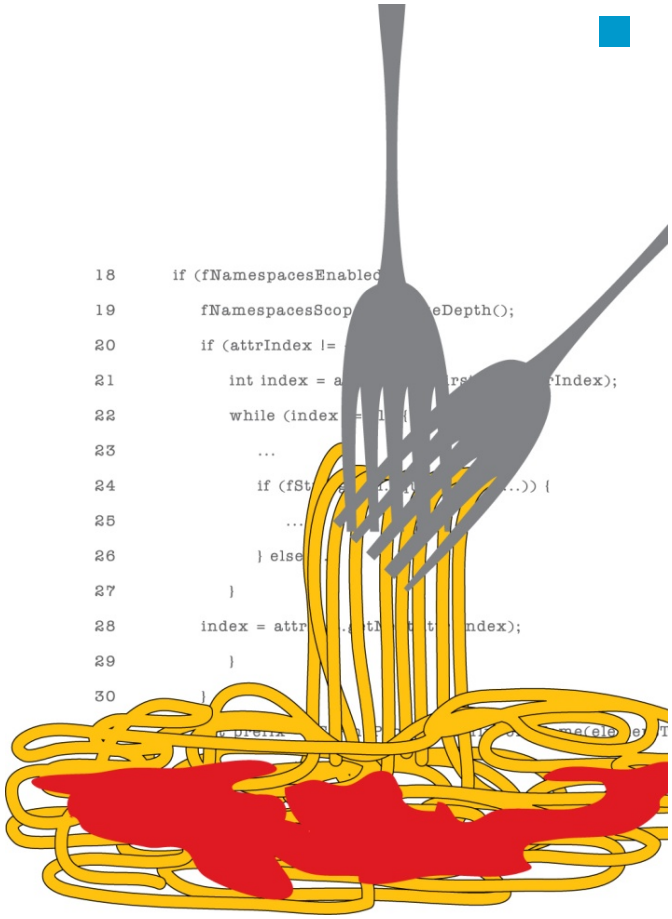
" Procedural-style design leads to one object with a lion's share of the responsibilities while most other objects only hold data or execute simple processes "

- Conception procédurale en programmation OO
- Large classe contrôleur
- Beaucoup d'attributs et méthodes avec une faible cohésion*
- Dépend de classes de données

** À quel point les méthodes sont étroitement liées aux attributs et aux méthodes de la classe.*

Défauts de conception

- 2 exemples d'anti-patrons

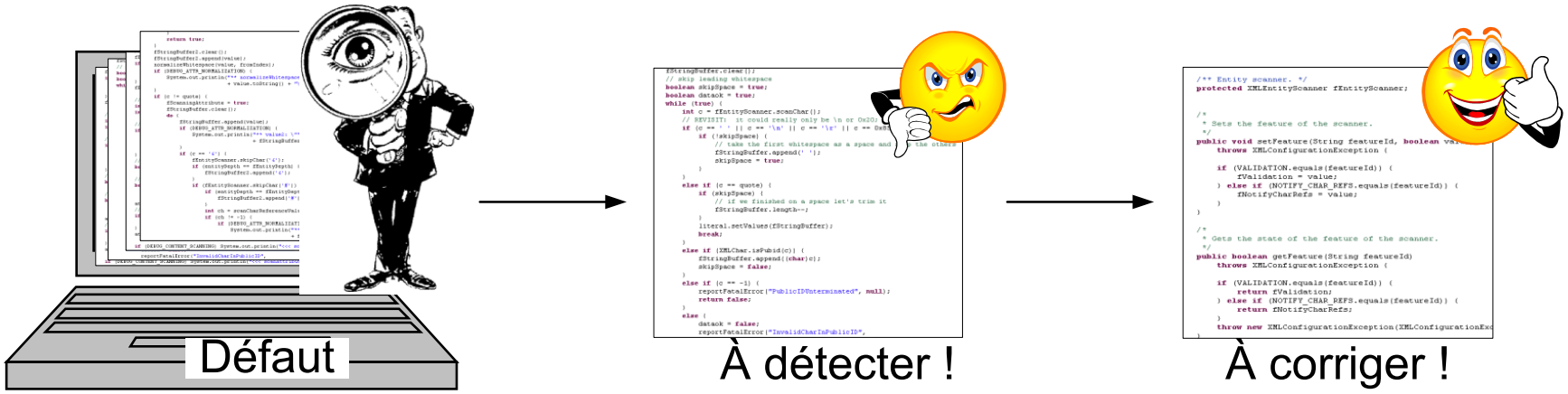


- Spaghetti Code

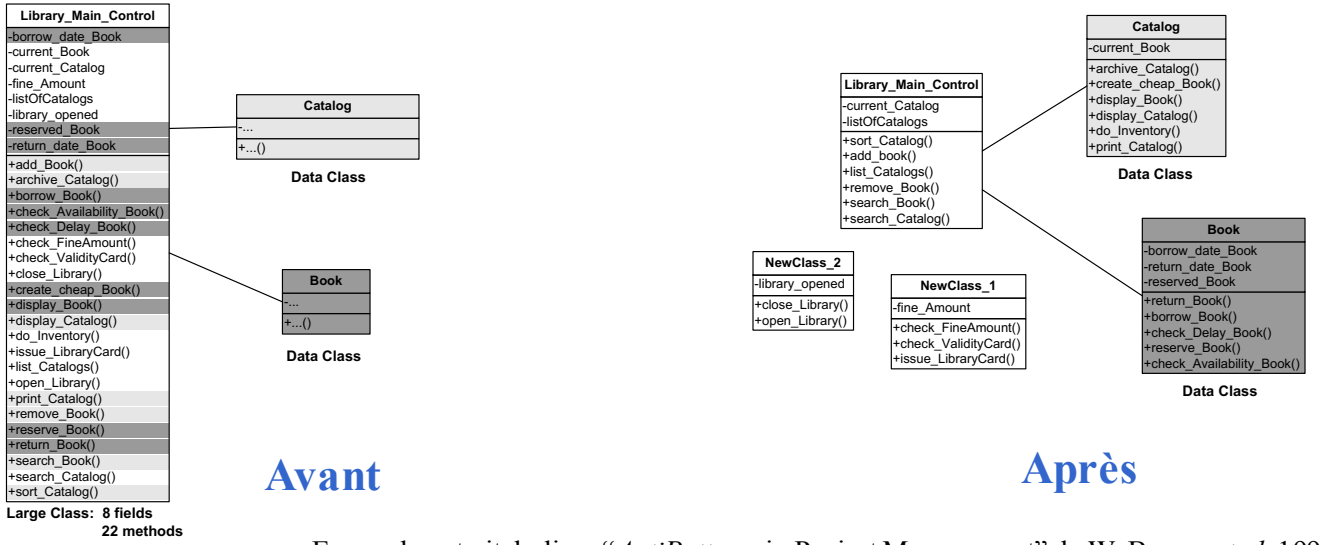
" Ad hoc software structure makes it difficult to extend and optimize code. "

  - Conception procédurale en programmation OO
  - Manque de structure : pas d'héritage, pas de réutilisation, pas de polymorphisme
  - Noms des classes suggèrent une programmation procédurale
  - Longues méthodes sans paramètres avec une faible cohésion
  - Utilisation excessive de variables globales

Défauts de conception



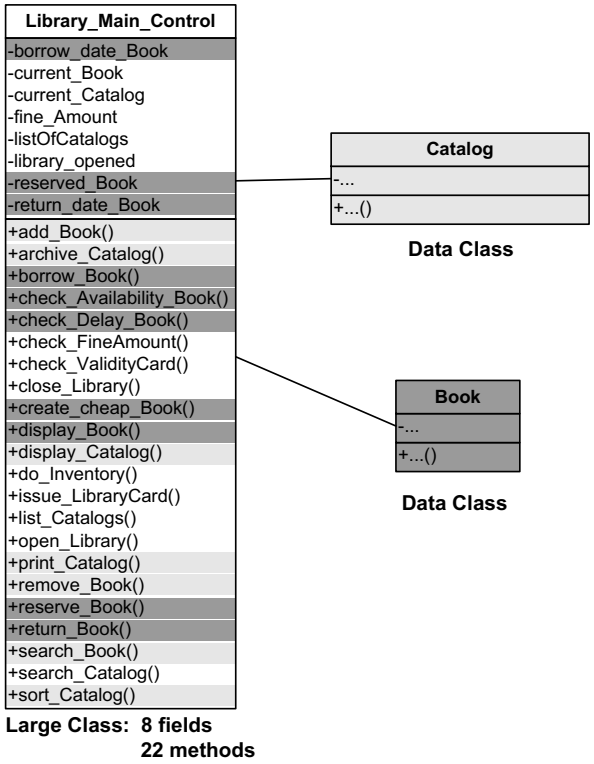
Anti-patron: Blob



Exemple extrait du livre "*AntiPatterns* in Project Management" de W. Brown *et al.* 1998

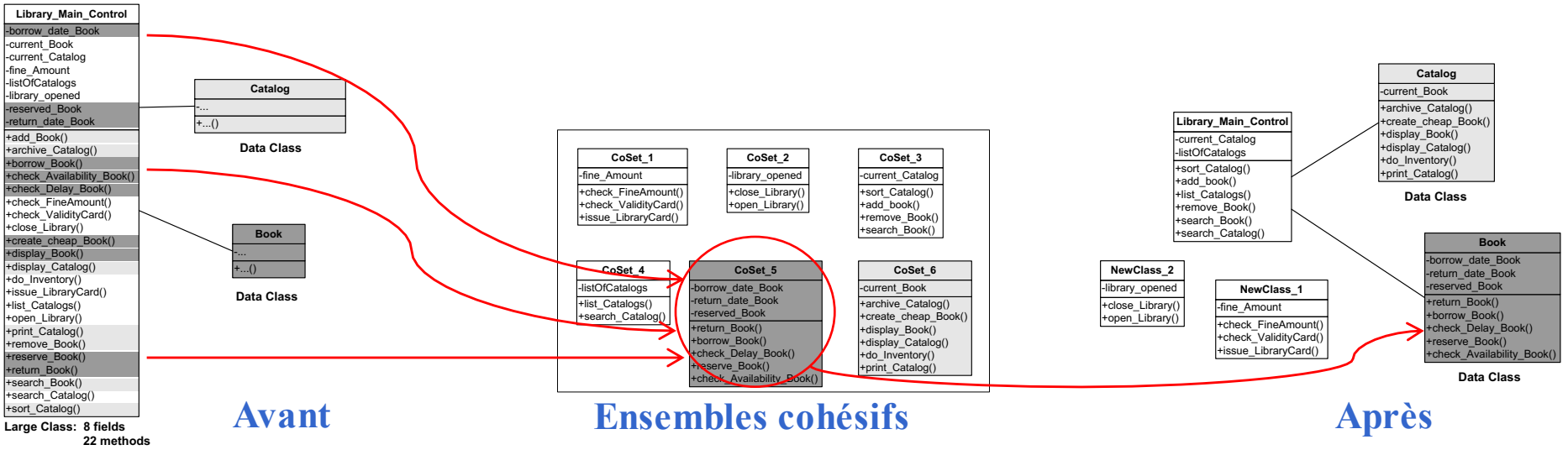
Détecter un anti-patron

- Blob
  - Identifier les larges classes
  - Identifier les classes de données

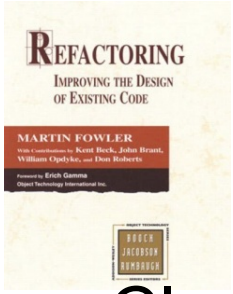


Corriger un anti-patron

- Blob
  - Identifier ou catégoriser les attributs et opérations liées
  - Rechercher des classes candidates pour les accueillir
  - Appliquer des techniques de conception objet (ex: héritage, délégation, patrons, etc.)

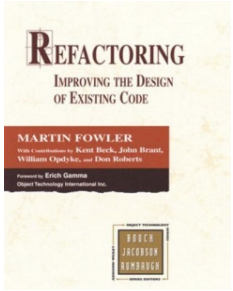


Mauvaises odeurs



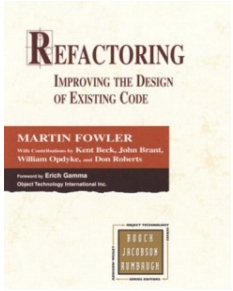
- Longue méthode
- Large classe
- Longue liste de paramètres
- Primitive obsession
- Groupe de données (Data clumps)
- Instructions Switch
- Champ temporaire
- Héritage refusé
- Classes alternatives avec des interfaces différentes
- Hiérarchies parallèles d'héritage
- Classe paresseuse
- Classe de données
- Code dupliqué
- Généralité spéculative
- Chaine de messages
- Middle man
- Feature envy
- Intimité inappropriée
- Divergent change
- Shotgun surgery
- Classe de librairie incomplète
- Commentaires

Mauvaises odeurs



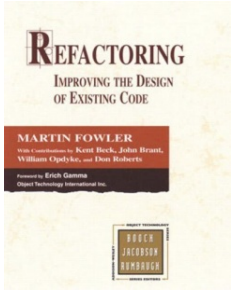
- Code dupliqué
  - Structure de code dupliquée à différents endroits
- Longue méthode
  - À décomposer pour viser la clarté et la facilité de maintenance
- Large classe
  - Classes qui essaient d'en faire trop. Présence de code dupliqué
- Longue liste de paramètres
  - Passer à la méthode juste ce dont elle a besoin

Mauvaises odeurs



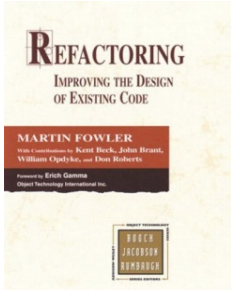
- Divergent Change
  - Si une classe est modifiée de différentes manières pour différentes raisons, ça vaut la peine de diviser la classe de sorte que chaque partie soit associée à un type de changement particulier
- Shotgun Surgery
  - Si un type de changement nécessite plusieurs petits changements de code dans différentes classes, tous ces bouts de code qui sont affectés devraient être mis ensemble dans une classe
- Feature Envy
  - Une méthode d'une classe est plus intéressée par les attributs d'une autre classe que celle de sa propre classe. Peut-être que placer la méthode dans cette autre classe serait plus appropriée
- Groupe de données
  - Le même groupe de données se retrouvent ensemble à plusieurs endroits: champs dans plusieurs classes, paramètres de méthodes, données locales. Peut-être qu'ils devraient être groupés ensemble dans une petite classe

Mauvaises odeurs



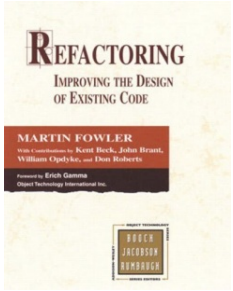
- Primitive Obsession
  - Parfois, plus intéressant de virer un type de données primitives vers une classe légère pour le rendre explicite et identifier les opérations à réaliser (ex: créer une classe date plutôt qu'utiliser un couple d'entiers)
- Instructions Switch
  - Tendent à créer de la duplication. Plusieurs instructions switch éparpillées à différents endroits. Utiliser des classes et du polymorphisme
- Hiérarchies parallèles d'héritage
  - Deux hiérarchies parallèles existent et un changement dans une classe de la hiérarchie nécessite des changements dans l'autre hiérarchie également
- Classe paresseuse
  - Les classes qui ne font pas assez de travail utile devrait être éliminé

Mauvaises odeurs



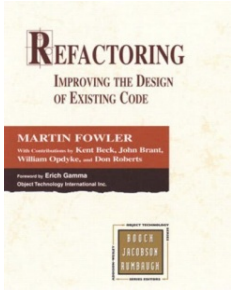
- Généralité spéculative
  - Code pas nécessaire a été créé pour anticiper les futurs changements du logiciel, rendant ainsi le logiciel plus complexe
- Champ temporaire
  - Confusion lorsque certaines variables d'une classe sont utilisées seulement occasionnellement
- Chaîne de messages
  - Une méthode envoie un message à un objet, lequel envoie un message à un autre objet, et ainsi de suite de sorte que la méthode originale devient couplée à tous ces objets
- Middle Man
  - La délégation est souvent utile, mais parfois ça peut aller loin. Si une classe agit comme mandataire, mais ne réalise pas de travail supplémentaire utile, il peut être utile de la supprimer de la hiérarchie

Mauvaises odeurs



- Intimité inappropriée
  - Classes qui dépensent trop de temps à fouiller dans les parties privées d'autres classes
- Classes alternatives avec des interfaces différentes
  - Classes qui font des choses similaires, mais ont des noms différents, devraient être modifiées pour partager un protocole commun
- Classe de librairie incomplète
  - Le logiciel utilise une librairie qui n'est pas complète, forçant les développeurs à étendre la fonctionnalité de la librairie
- Classe de données
  - Classes qui ont juste des champs de données et des méthodes d'accès, mais pas de réel comportement. Si les données sont publiques, rendez-les privées!

Mauvaises odeurs



- Héritage refusé
  - Si une sous-classe ne veut pas ou n'a pas besoin de tout le comportement de sa base classe, alors peut-être la hiérarchie de classes est fausse
- Commentaires
  - Si les commentaires sont présents dans le code car le code est mauvais, alors améliorer le code

Trouver le défaut

```
class OwnershipTest…
   private void createUserInGroup() {
      GroupManager groupManager = new GroupManager();
      Group group = groupManager.create(TEST_GROUP, false,
                                                GroupProfile.UNLIMITED_LICENSES, "",
                                                      GroupProfile.ONE_YEAR, null);
      user = userManager.create(USER_NAME, group, USER_NAME, "joshua", USER_NAME,
                                    LANGUAGE, false, false, new Date(), "blah", new Date());
}
```

Code extrait de Industrial Logic, Inc.

Trouver le défaut

```
class OwnershipTest…
   private void createUserInGroup() {
      GroupManager groupManager = new GroupManager();
      Group group = groupManager.create(TEST_GROUP, false,
                                                GroupProfile.UNLIMITED_LICENSES, "",
                                                      GroupProfile.ONE_YEAR, null);
      user = userManager.create(USER_NAME, group, USER_NAME, "joshua", USER_NAME,
                                LANGUAGE, false, false, new Date(), "blah", new Date());
}
```

Longue liste de paramètres

Code extrait de Industrial Logic, Inc.

Trouver le défaut

```
public class Phone {

  private final String unformattedNumber;

  public Phone(String unformattedNumber) {
     this.unformattedNumber = unformattedNumber;
  }

  public String getAreaCode() {
     return unformattedNumber.substring(0,3);
  }

  public String getPrefix() {
     return unformattedNumber.substring(3,6);
  }

  public String getNumber() {
     return unformattedNumber.substring(6,10);
  }
}

public class Customer…
  private Phone mobilePhone;

  public String getMobilePhoneNumber() {
    return "(" + mobilePhone.getAreaCode() + ") "
               + mobilePhone.getPrefix() + "-"
               + mobilePhone.getNumber();
}
```

Code extrait de Industrial Logic, Inc.

Trouver le défaut

```
public class Phone {

  private final String unformattedNumber;

  public Phone(String unformattedNumber) {
     this.unformattedNumber = unformattedNumber;
  }

  public String getAreaCode() {
     return unformattedNumber.substring(0,3);
  }

  public String getPrefix() {
     return unformattedNumber.substring(3,6);
  }

  public String getNumber() {
     return unformattedNumber.substring(6,10);
  }
}

public class Customer…
  private Phone mobilePhone;

  public String getMobilePhoneNumber() {
    return "(" + mobilePhone.getAreaCode() + ") "
               + mobilePhone.getPrefix() + "-"
               + mobilePhone.getNumber();
}
```

Feature Envy

Customer va rechercher dans les données de Phone

Code extrait de Industrial Logic, Inc.

Trouver le défaut

```
public class Phone {

  private final String unformattedNumber;

  public Phone(String unformattedNumber) {
     this.unformattedNumber = unformattedNumber;
  }

  public String getAreaCode() {
     return unformattedNumber.substring(0,3);
  }

  public String getPrefix() {
     return unformattedNumber.substring(3,6);
  }

  public String getNumber() {
     return unformattedNumber.substring(6,10);
  }

  public String toFormattedString() {
     return "(" + getAreaCode() + ") " + getPrefix() + "-" + getNumber(); }
}

public class Customer…
  private Phone mobilePhone;

  public String getMobilePhoneNumber() {
    return mobilePhone.toFormattedString();
  }
```

Sans Feature Envy

Customer compte sur Phone pour faire le formatage

Code extrait de Industrial Logic, Inc. 22

Trouver le défaut

```
public abstract class AbstractCollection implements collection…
   public void addAll(AbstractCollection c) {
      if(c instanceof Set) {
       Set s = (Set)c;
       for(int i=0; i<s.size();i++){
              if(!contains(s.getElementAt(i))){
                     add(s.getElementAt(i));
              }
       }
      } else if(c instanceof List) {
              List l = (List)c;
              for(int i=0;i<l.size();i++){
                     if(!contains(l.get(i))){
                            add(l.get(i));
                     }
              }
       }
```

Code extrait de Industrial Logic, Inc.

Trouver le défaut

```
public abstract class AbstractCollection implements collection…
   public void addAll(AbstractCollection c) {
      if(c instanceof Set) {
       Set s = (Set)c;
       for(int i=0; i<s.size();i++){
              if(!contains(s.getElementAt(i))){
                     add(s.getElementAt(i));
              }
       }
      } else if(c instanceof List) {
              List l = (List)c;
              for(int i=0;i<l.size();i++){
                     if(!contains(l.get(i))){
                            add(l.get(i));
                     }
              }
       }
```

Instruction Switch

Classes alternatives avec interfaces différentes

Code dupliqué

Code extrait de Industrial Logic, Inc.

Trouver le défaut

```
public abstract class AbstractCollection…
    public void add(Object element) {
    }


public class Map extends AbstractCollection…
    // Do nothing because user must input key and value
    public void add(Object element) {
    }
```

Code extrait de Industrial Logic, Inc.

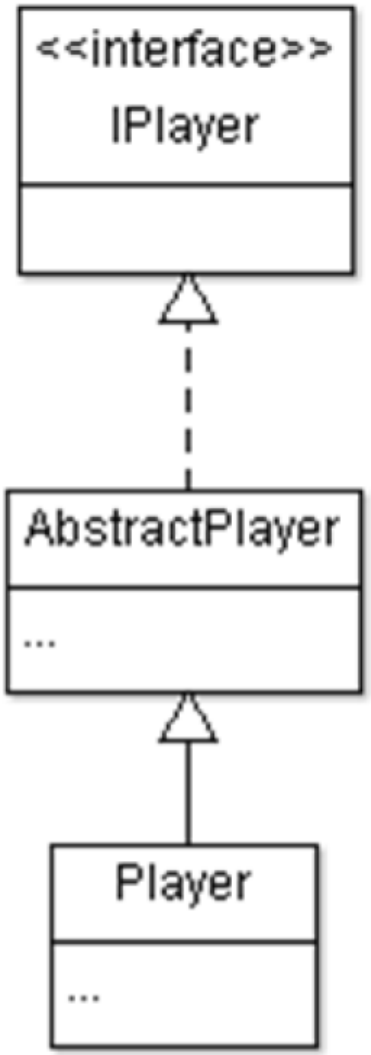
Trouver le défaut

```
public abstract class AbstractCollection…
    public void add(Object element) {
    }


public class Map extends AbstractCollection…
    // Do nothing because user must input key and value
    public void add(Object element) {
    }
```

Héritage refusé

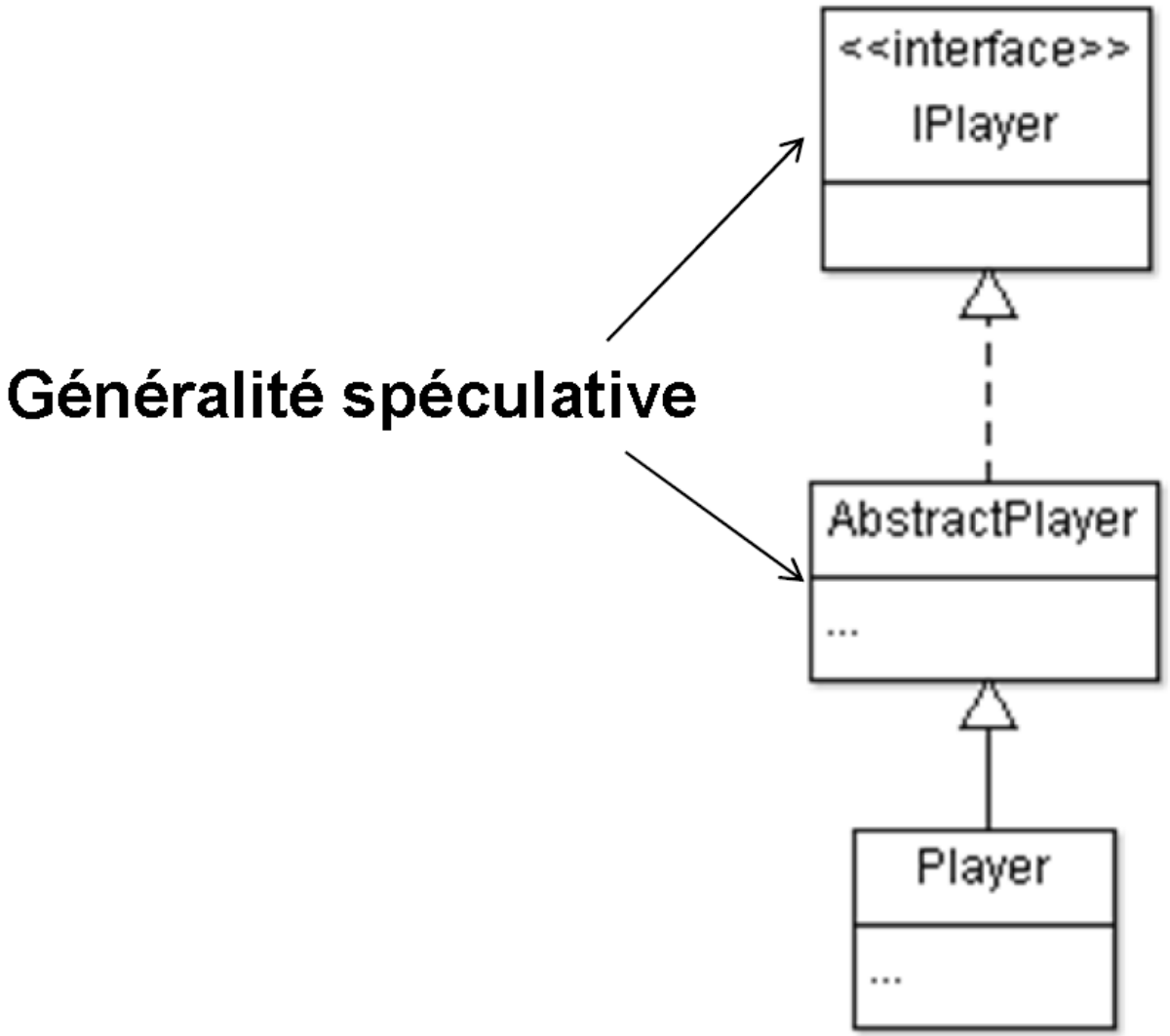
Code extrait de Industrial Logic, Inc.

Trouver le défaut



Code extrait de Industrial Logic, Inc.

Trouver le défaut



Code extrait de Industrial Logic, Inc.

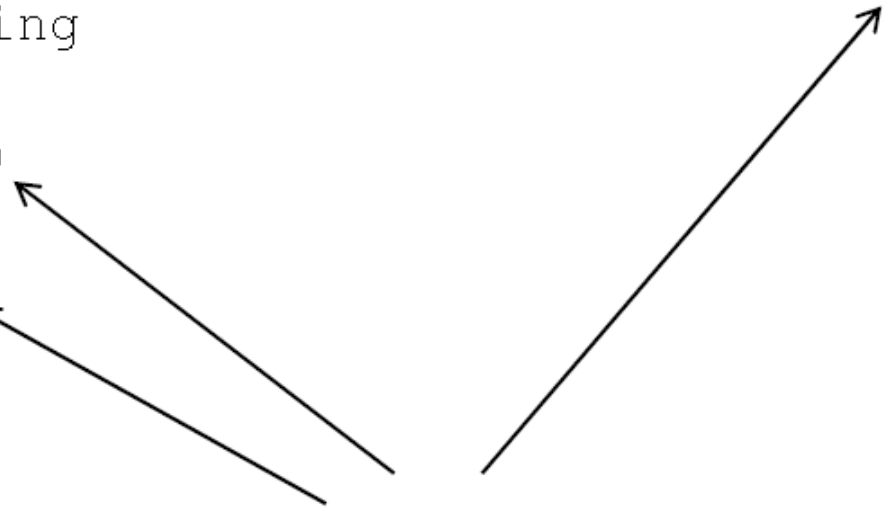
Trouver le défaut

```
if ( something.indexOf(``substring'') != -1 )
    // do something

if(result==null)
    ...
    return null;
```

Code extrait de Industrial Logic, Inc.

Trouver le défaut

```
if ( something.indexOf(''substring'') != -1 )
    // do something

if(result==null)
    ...
    return null;
```



Primitive Obesession

Solution:
- isEmpty()
- Throws exceptions

Code extrait de Industrial Logic, Inc.

Trouver le défaut

- Vidéo : Smells in Map
  - https://elearning.industriallogic.com/gh/submit?Action=PageAction&album=codeSmellsAndRefactoring&path=recognizingSmells/smellsInLegacyCodeAScavengerHunt/map&devLanguage=Java

Détecter les défauts

- Métriques logicielles Chidamber et Kemerer (1994)
  - Mesures utilisées pour quantifier le logiciel, les ressources et/ou le processus de développement logiciel
  - Exemples ? lignes de code (LOC), complexité cyclomatique (CC), cohésion, couplage, profondeur d'héritage (DIT), nombre de méthodes / attributs / paramètres, etc.
- Analyses statiques et dynamiques
- Analyses lexicales

Détecter les défauts

- Long method
  - LOC et CC
- Long parameter lists
  - Nombre de paramètres
- Large class
  - Nombre d'attributs et de méthodes
- Feature envy
  - Couplage
- Inappropriate intimacy
  - Couplage

Détecter les défauts

- Spaghetti Code

  " Ad hoc software structure makes it difficult to extend and optimize code. "

  – Conception procédurale en programmation OO

  – Manque de structure : pas d'héritage, pas de réutilisation, pas de polymorphisme

  – Noms des classes suggèrent une programmation procédurale

  – Longues méthodes sans paramètres avec une faible cohésion

  – Utilisation excessive de variables globales

Détecter les défauts

- Spaghetti Code

“ Ad hoc software structure makes it difficult to extend and optimize code. ”

– C

– M
ré

– N
pr

– L
fa

– U

```
CODESMELL  define  LongMethod     as  METRIC  LOC_METHOD with  VERY HIGH and 10.0 ;
CODESMELL  define  NoParameter    as  METRIC  NMNOPARAM  with  VERY HIGH and 5.0 ;
CODESMELL  define  NoInheritance  as  METRIC  DIT        with  1 and 0.0 ;
CODESMELL  define  NoPolymorphism as  STRUC   NO_POLYMORPHISM ;
CODESMELL  define  ProceduralName as  LEXIC   CLASS_NAME with  (Make, Create, Exec) ;
CODESMELL  define  GlobalVariable as  STRUC   USE_GLOBAL_VARIABLE ;

ANTIPATTERN define SpaghettiCode  as {
        ((LongMethod INTER NoParameter) INTER (NoInheritance UNION NoPolymorphism))
   INTER
        (ProceduralName UNION UseGlobalVariable) } ;
```

Mesure de la complexité

- Complexité cyclomatique de McCabe
  - Se calcule à partir d'un graphe de flot de contrôle

Structure du flot de contrôle

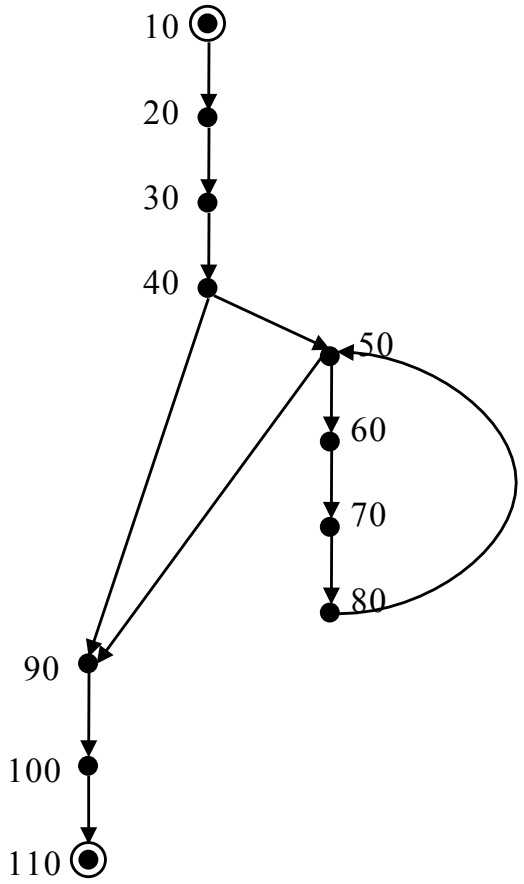
- De nombreux travaux sur les métriques portent sur la structure du flot de contrôle
- Le graphe de flot de contrôle est un graphe orienté
  - Nœud = instruction du programme
  - Arc = flot de contrôle entre instructions

  - Degré d'entrée d'un nœud = nombre d'arcs incidents
  - Degré de sortie d'un nœud = nombre d'arcs sortants
  - Nœud procédure = un nœud de degré de sortie 1
  - Nœud prédicat = un nœud de degré de sortie > 1
  - Chemin = une séquence d'arcs consécutifs
  - Chemin simple = chemin sans répétition

Structure du flot de contrôle

- Exemple de graphe de flot de contrôle

```
10  Lire p
20  Lire e
30  Cal := 1
40  Si e ≠ 0 Alors
50  Tant Que e > 0 Faire
60  cal := cal × p
70  e := e - 1
80  Fin Faire
90  Fin Si
100 Écrire cal
110 Fin
```

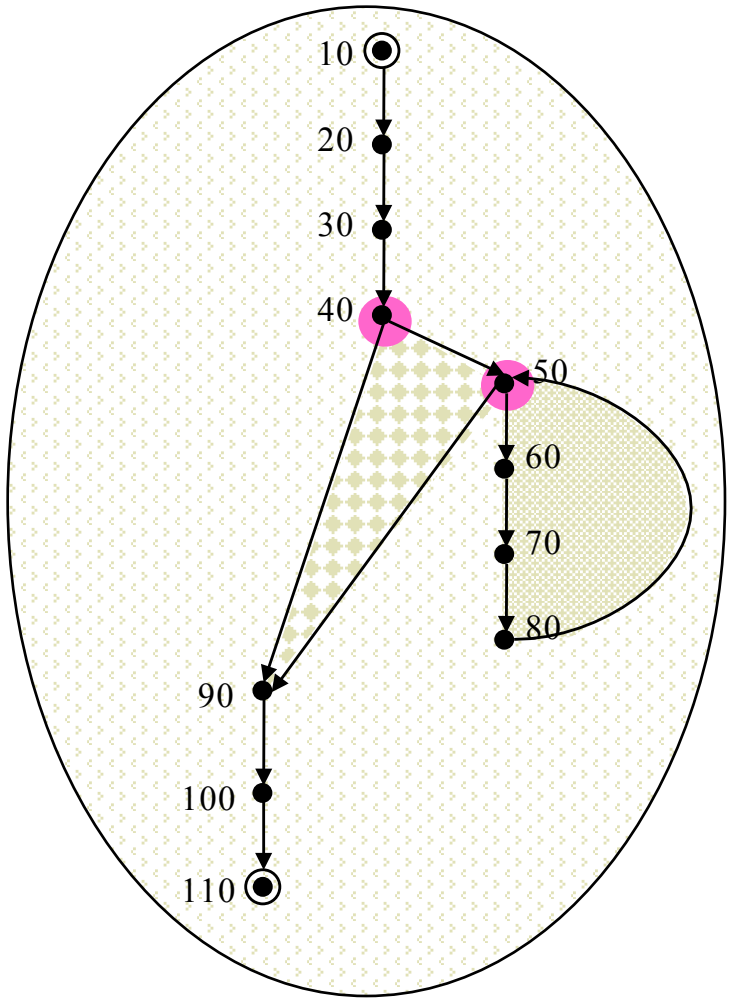


Mesure de la complexité

- Complexité cyclomatique de McCabe
  - Pour un graphe de flot de contrôle F comportant *n* nœuds (dont *d* nœuds prédicats) et *e* arcs, il existe trois façons de mesurer la complexité
    - $V(F) = e - n + 2$
    - $V(F) = 1 + d$
    - $V(F) = r$, où *r* est le nombre de régions de F

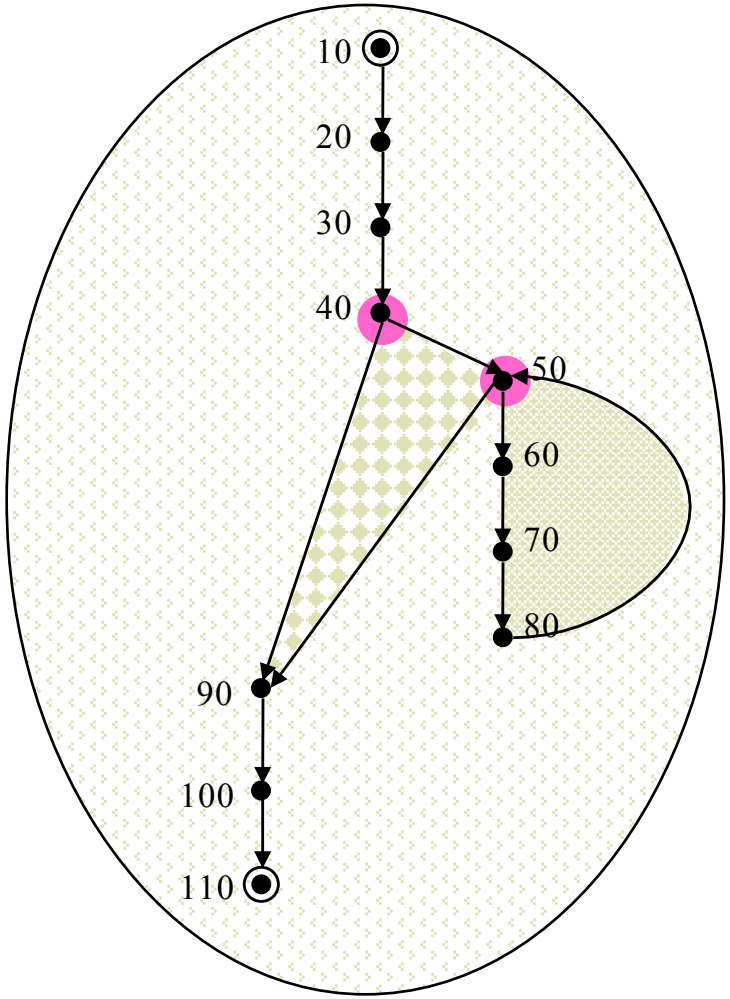
Mesure de la complexité

- Complexité cyclomatique de McCabe
  - Pour un graphe de flot de contrôle F comportant *n* nœuds (dont *d* nœuds prédicats) et *e* arcs, il existe trois façons de mesurer la complexité
    - $V(F) = e - n + 2$
    - $V(F) = 1 + d$
    - $V(F) = r$, où *r* est le nombre de régions de F



Mesure de la complexité

- Complexité cyclomatique de McCabe
  - *V(F)* = 12 – 11 + 2
  - *V(F)* = 1 + 2
  - *V(F)* = 3



Les métriques logicielles

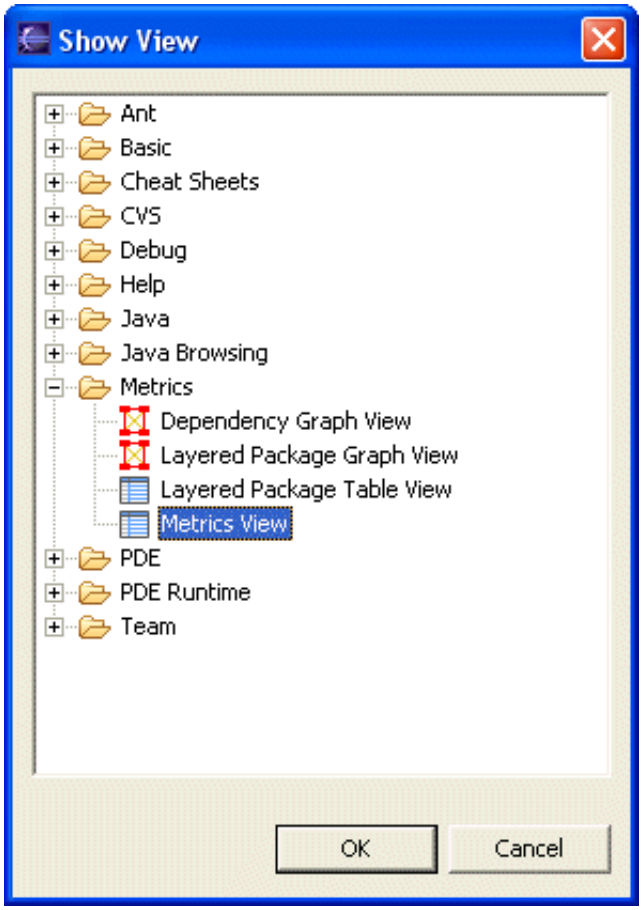
- Coupling Between Objects (CBO)
  - Nombre d'autres classes auxquelles une classe est couplée, deux classes sont dites couplées si une méthode de l'une utilise une méthode (ou un attribut ☹) de l'autre
- Depth of Inheritance Tree (DIT)
  - Profondeur de la classe dans l'arbre d'héritage
  - En cas d'héritage multiple, DIT est égale à la profondeur maximale depuis la classe racine jusqu'à la classe mesurée

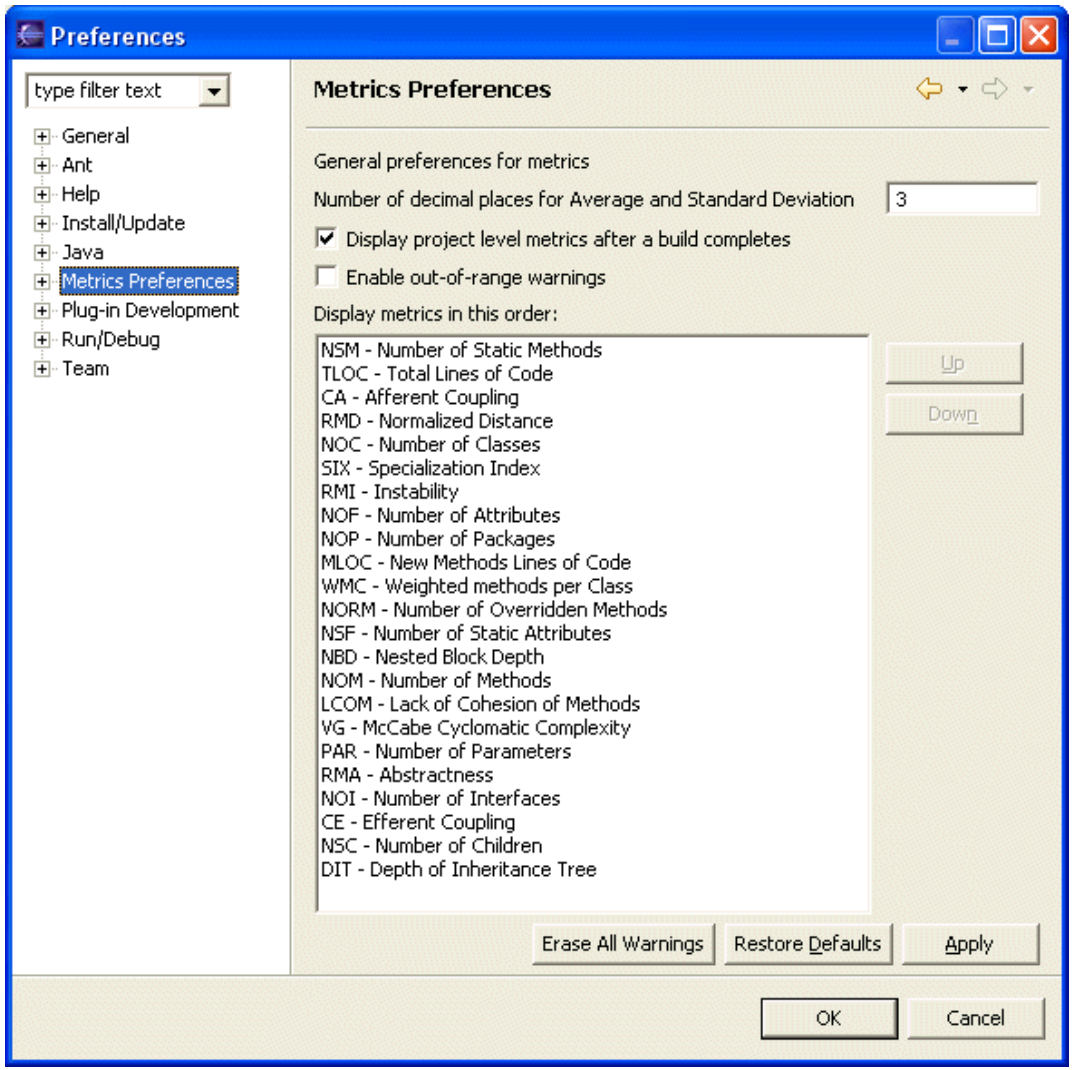
Les métriques logicielles

- Number Of Children (NOC)
  - Nombre de descendants immédiats de la classe dans la hiérarchie de classes
- Lack of COhesion in Methods (LCOM)
  - Degré qui définit à quel point les méthodes sont étroitement liées aux attributs et aux méthodes de la classe
  - LCOM compte le nombre de paires de méthodes qui n'accèdent pas aux même attributs moins le nombre de paires de méthodes qui accèdent aux même attributs (= 0 si négative)

Outils

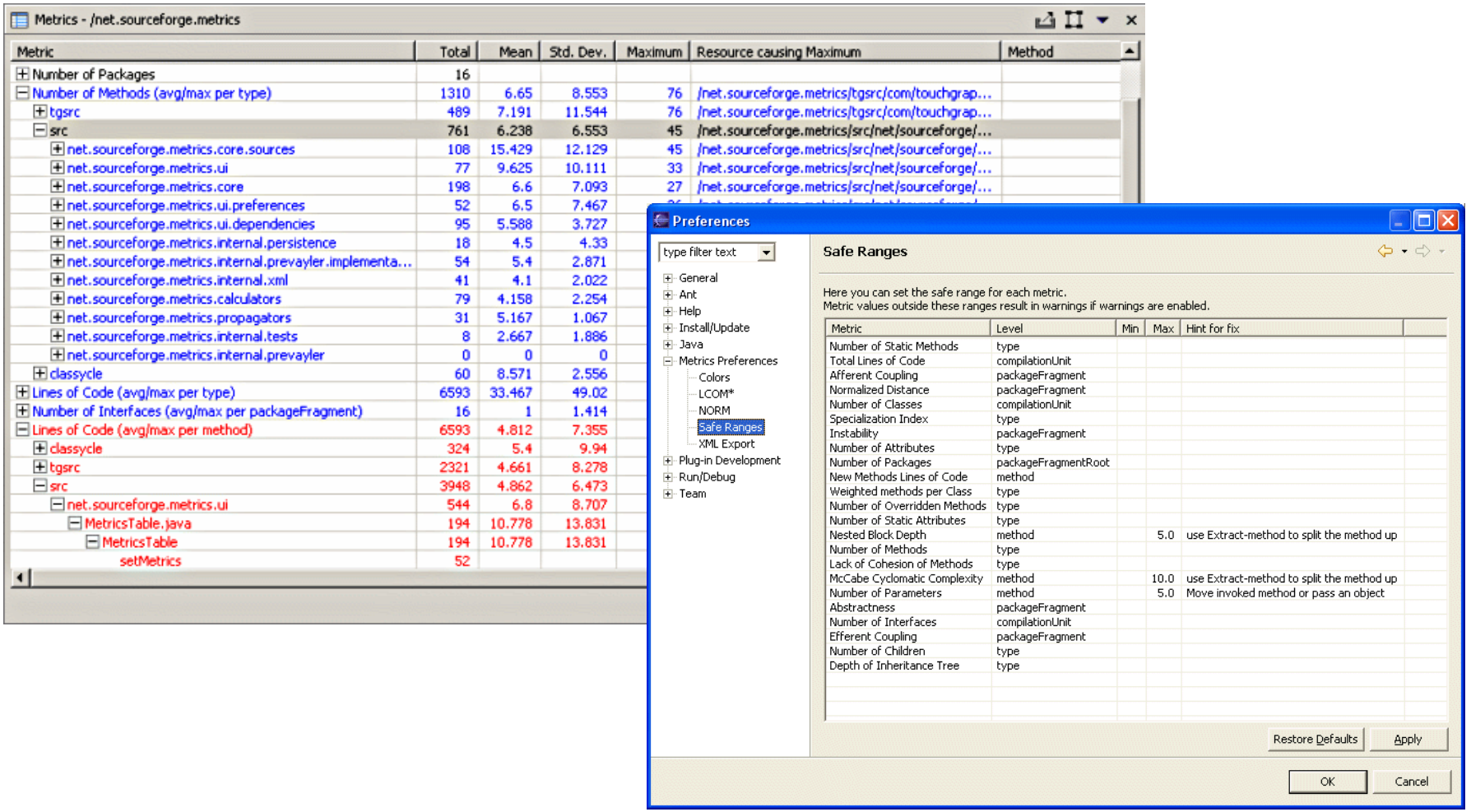
- Eclipse Metrics Plug-in http://metrics.sourceforge.net/





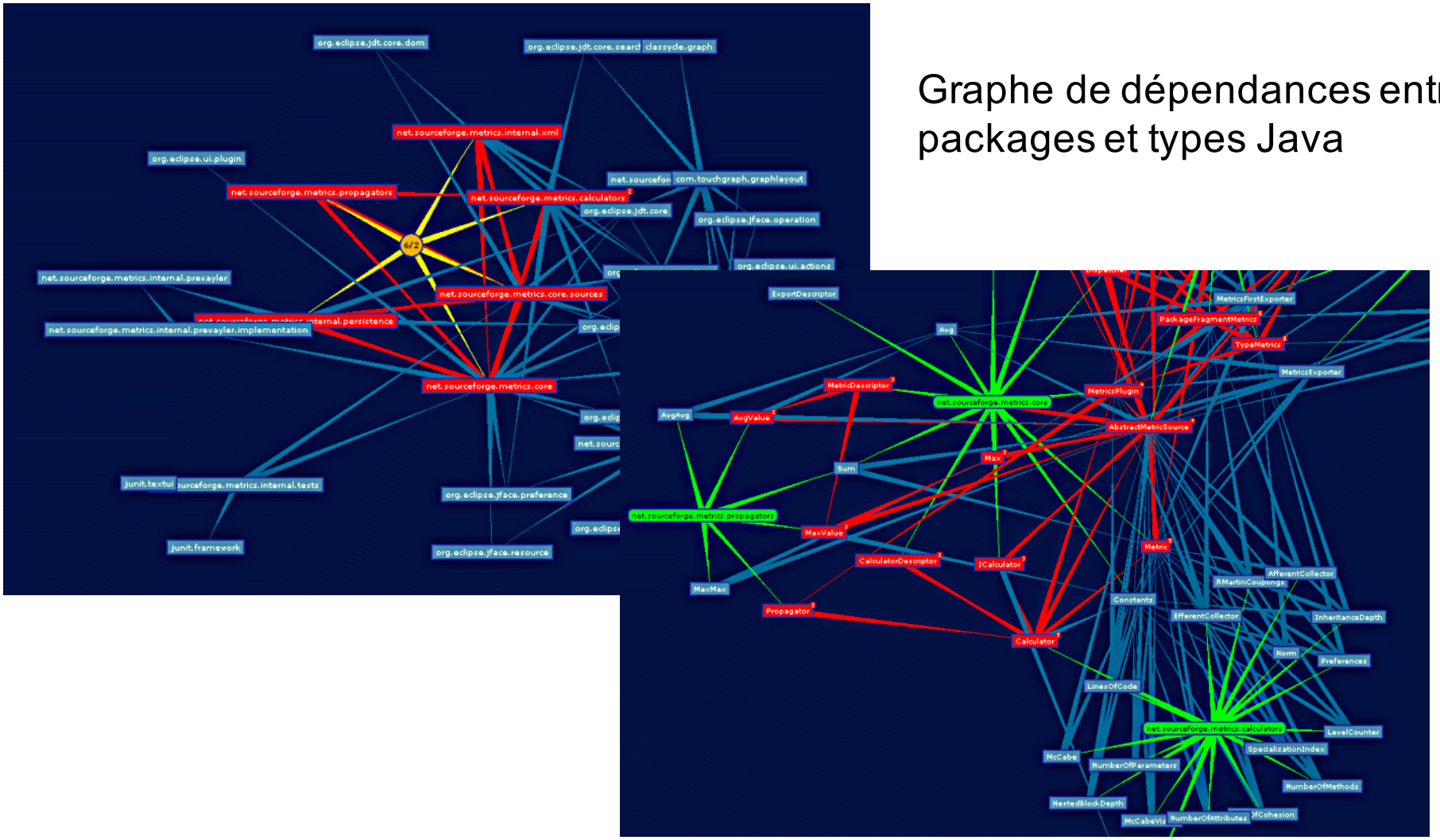
Outils

- Eclipse Metrics Plug-in http://metrics.sourceforge.net/



Outils

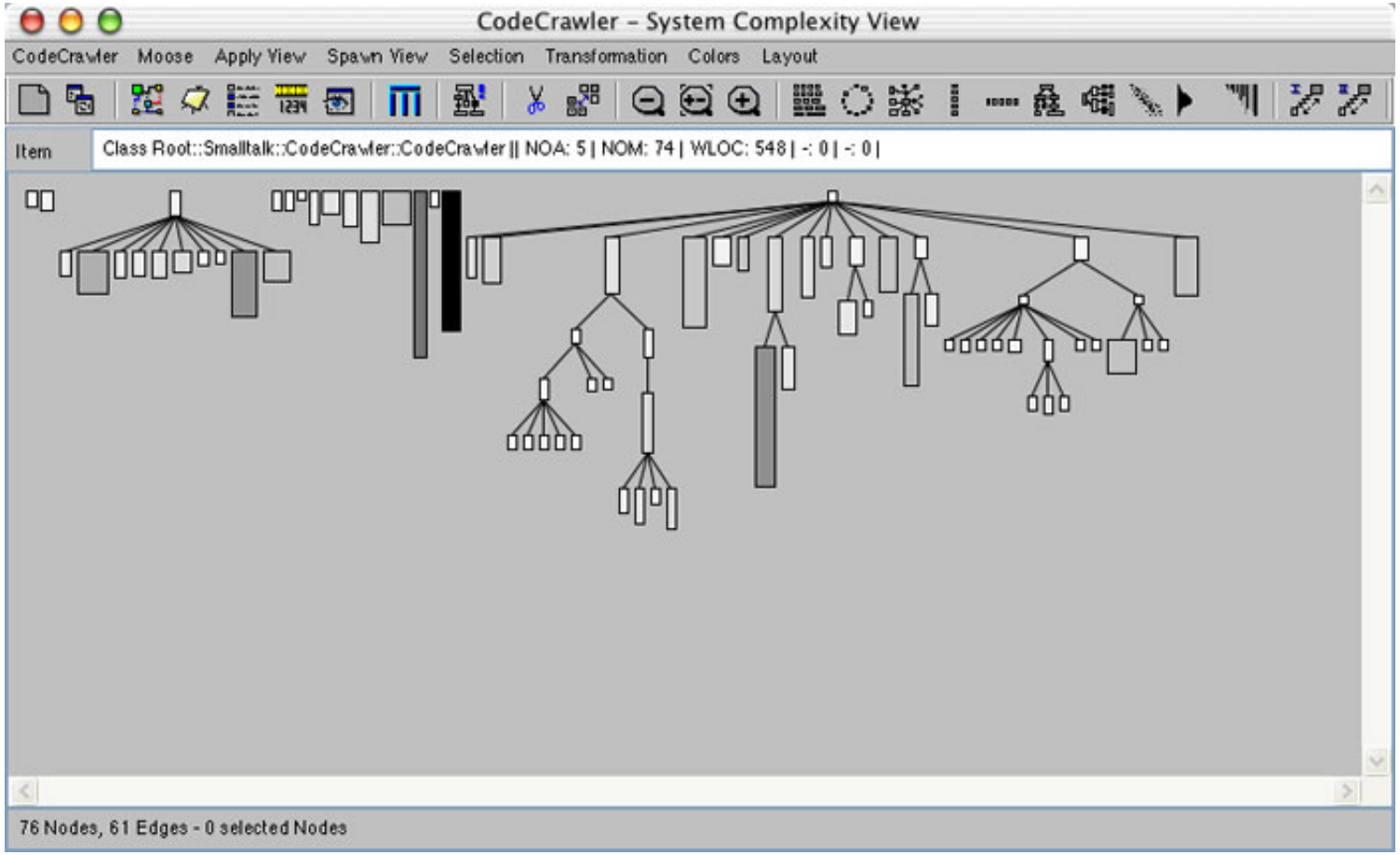
- Eclipse Metrics Plug-in http://metrics.sourceforge.net/



Graphe de dépendances entre packages et types Java

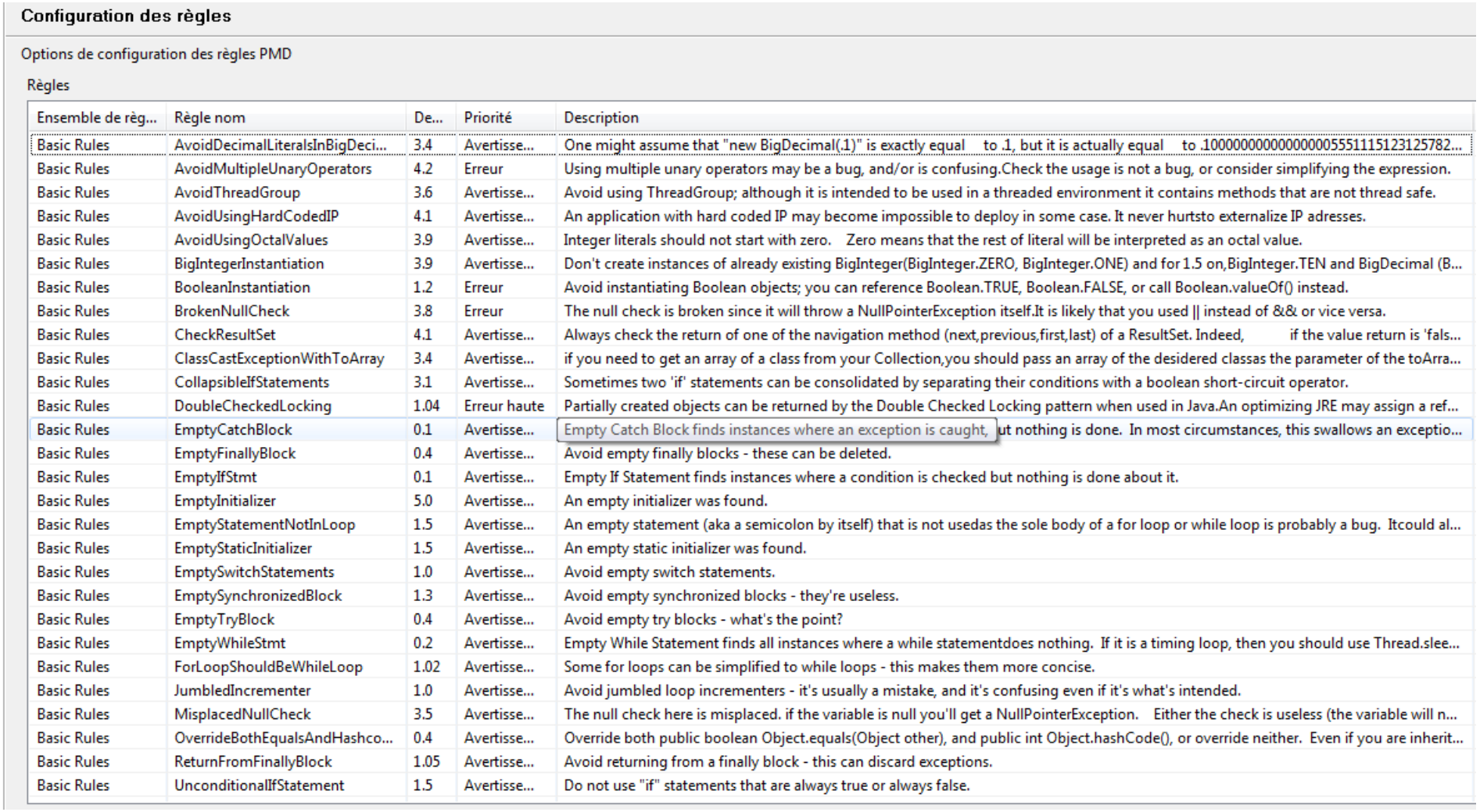
Outils

- CodeCrawler



Outils

- PMD http://pmd.sourceforge.net/



Configuration des règles

Options de configuration des règles PMD

Règles

Ensemble de règ...	Règle nom	De...	Priorité	Description
Basic Rules	AvoidDecimalLiteralsInBigDeci...	3.4	Avertisse...	One might assume that "new BigDecimal(.1)" is exactly equal to .1, but it is actually equal to .1000000000000000055511151231257782...
Basic Rules	AvoidMultipleUnaryOperators	4.2	Erreur	Using multiple unary operators may be a bug, and/or is confusing.Check the usage is not a bug, or consider simplifying the expression.
Basic Rules	AvoidThreadGroup	3.6	Avertisse...	Avoid using ThreadGroup; although it is intended to be used in a threaded environment it contains methods that are not thread safe.
Basic Rules	AvoidUsingHardCodedIP	4.1	Avertisse...	An application with hard coded IP may become impossible to deploy in some case. It never hurtsto externalize IP adresses.
Basic Rules	AvoidUsingOctalValues	3.9	Avertisse...	Integer literals should not start with zero. Zero means that the rest of literal will be interpreted as an octal value.
Basic Rules	BigIntegerInstantiation	3.9	Avertisse...	Don't create instances of already existing BigInteger(BigInteger.ZERO, BigInteger.ONE) and for 1.5 on,BigInteger.TEN and BigDecimal (B...
Basic Rules	BooleanInstantiation	1.2	Erreur	Avoid instantiating Boolean objects; you can reference Boolean.TRUE, Boolean.FALSE, or call Boolean.valueOf() instead.
Basic Rules	BrokenNullCheck	3.8	Erreur	The null check is broken since it will throw a NullPointerException itself.It is likely that you used || instead of && or vice versa.
Basic Rules	CheckResultSet	4.1	Avertisse...	Always check the return of one of the navigation method (next,previous,first,last) of a ResultSet. Indeed, if the value return is 'fals...
Basic Rules	ClassCastExceptionWithToArray	3.4	Avertisse...	if you need to get an array of a class from your Collection,you should pass an array of the desidered classas the parameter of the toArra...
Basic Rules	CollapsibleIfStatements	3.1	Avertisse...	Sometimes two 'if' statements can be consolidated by separating their conditions with a boolean short-circuit operator.
Basic Rules	DoubleCheckedLocking	1.04	Erreur haute	Partially created objects can be returned by the Double Checked Locking pattern when used in Java.An optimizing JRE may assign a ref...
Basic Rules	EmptyCatchBlock	0.1	Avertisse...	Empty Catch Block finds instances where an exception is caught, ut nothing is done. In most circumstances, this swallows an exceptio...
Basic Rules	EmptyFinallyBlock	0.4	Avertisse...	Avoid empty finally blocks - these can be deleted.
Basic Rules	EmptyIfStmt	0.1	Avertisse...	Empty If Statement finds instances where a condition is checked but nothing is done about it.
Basic Rules	EmptyInitializer	5.0	Avertisse...	An empty initializer was found.
Basic Rules	EmptyStatementNotInLoop	1.5	Avertisse...	An empty statement (aka a semicolon by itself) that is not usedas the sole body of a for loop or while loop is probably a bug. Itcould al...
Basic Rules	EmptyStaticInitializer	1.5	Avertisse...	An empty static initializer was found.
Basic Rules	EmptySwitchStatements	1.0	Avertisse...	Avoid empty switch statements.
Basic Rules	EmptySynchronizedBlock	1.3	Avertisse...	Avoid empty synchronized blocks - they're useless.
Basic Rules	EmptyTryBlock	0.4	Avertisse...	Avoid empty try blocks - what's the point?
Basic Rules	EmptyWhileStmt	0.2	Avertisse...	Empty While Statement finds all instances where a while statementdoes nothing. If it is a timing loop, then you should use Thread.slee...
Basic Rules	ForLoopShouldBeWhileLoop	1.02	Avertisse...	Some for loops can be simplified to while loops - this makes them more concise.
Basic Rules	JumbledIncrementer	1.0	Avertisse...	Avoid jumbled loop incrementers - it's usually a mistake, and it's confusing even if it's what's intended.
Basic Rules	MisplacedNullCheck	3.5	Avertisse...	The null check here is misplaced. if the variable is null you'll get a NullPointerException. Either the check is useless (the variable will n...
Basic Rules	OverrideBothEqualsAndHashco...	0.4	Avertisse...	Override both public boolean Object.equals(Object other), and public int Object.hashCode(), or override neither. Even if you are inherit...
Basic Rules	ReturnFromFinallyBlock	1.05	Avertisse...	Avoid returning from a finally block - this can discard exceptions.
Basic Rules	UnconditionalIfStatement	1.5	Avertisse...	Do not use "if" statements that are always true or always false.

Outils

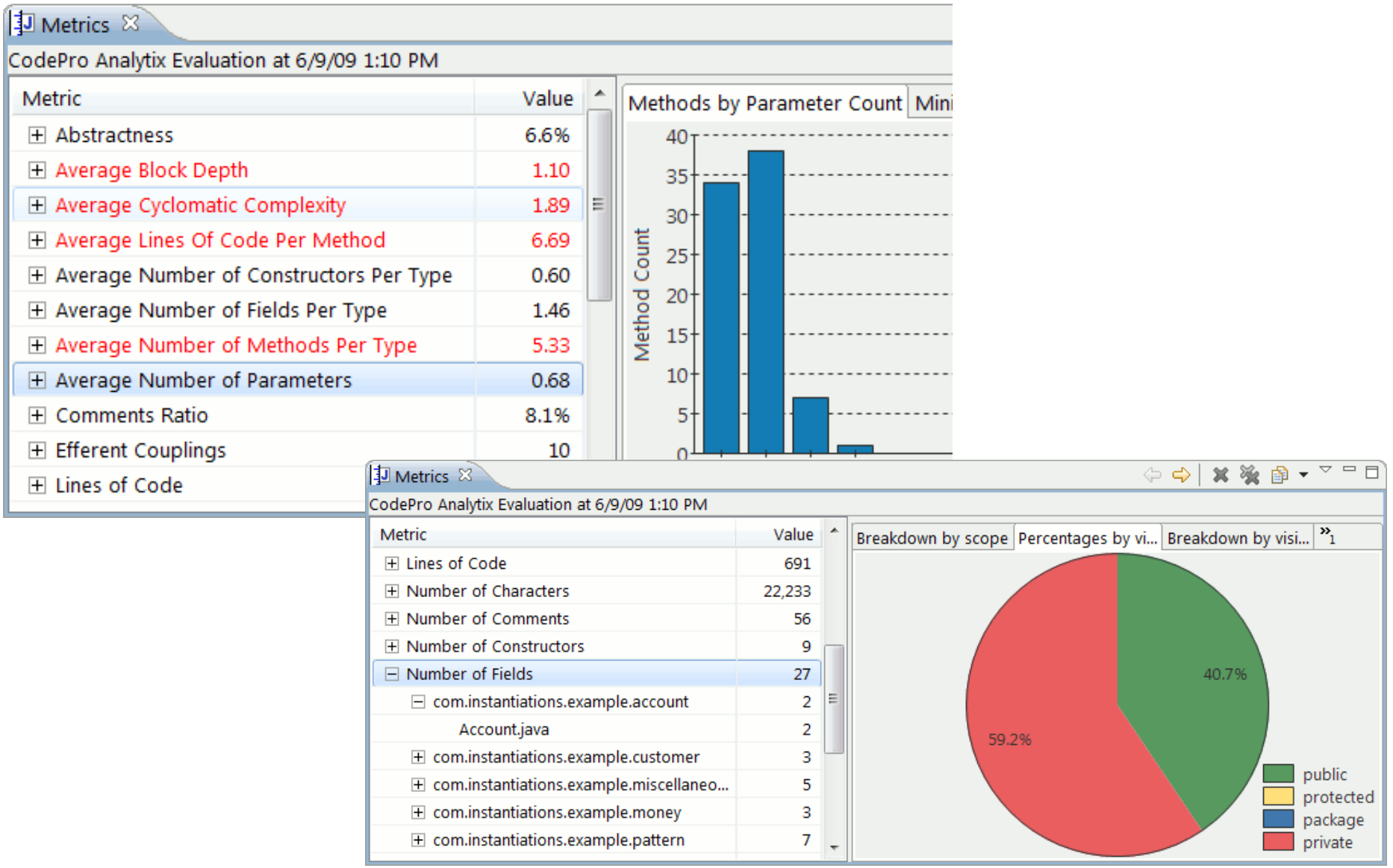
- PMD http://pmd.sourceforge.net/

Synthèse des alertes PMD | Flots de données | Alertes PMD

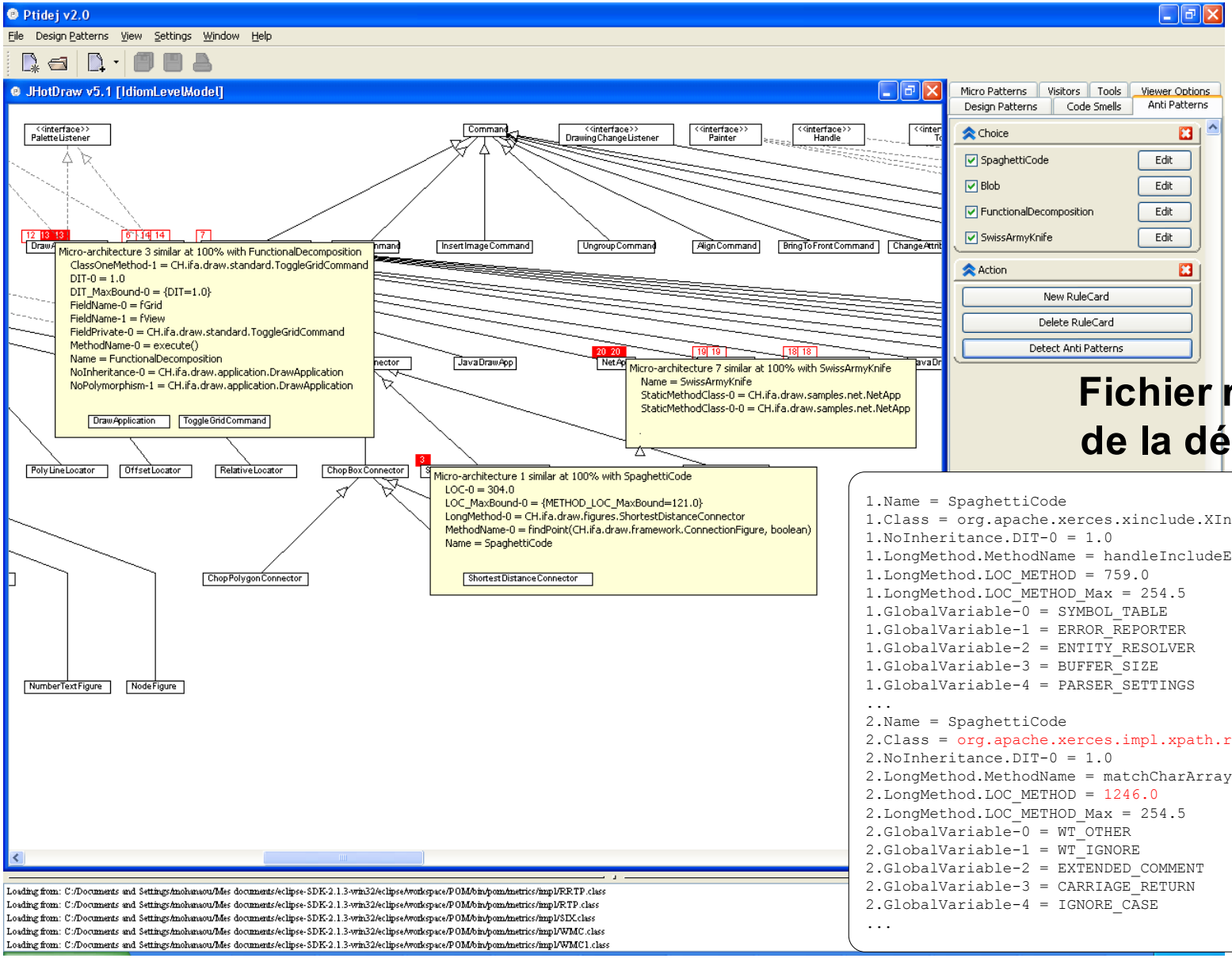
Elément	# Violations	# Violations/L...	# Violatio...	Projet
fr.irisa.triskell.eclipse.console	3	10.5 / 1000	0.08	fr.irisa.triskell.eclipse.util
IOConsole.java	1	13.9 / 1000	0.08	fr.irisa.triskell.eclipse.util
EmptyMethodInAbstractClassShouldBeAbstract	1	13.9 / 1000	0.08	fr.irisa.triskell.eclipse.util
EclipseConsole.java	2	12.0 / 1000	0.14	fr.irisa.triskell.eclipse.util
ConstructorCallsOverridableMethod	2	12.0 / 1000	0.14	fr.irisa.triskell.eclipse.util
fr.irisa.triskell.eclipse.resources	2	4.1 / 1000	0.05	fr.irisa.triskell.eclipse.util
ResourceHelper.java	2	9.1 / 1000	0.12	fr.irisa.triskell.eclipse.util
VariableNamingConventions	2	9.1 / 1000	0.12	fr.irisa.triskell.eclipse.util
org.kermeta.customidentity	2	6.8 / 1000	0.08	fr.irisa.triskell.eclipse.util
CustomHashtable.java	2	7.0 / 1000	0.10	fr.irisa.triskell.eclipse.util
ConstructorCallsOverridableMethod	1	3.5 / 1000	0.05	fr.irisa.triskell.eclipse.util
AvoidThrowingNullPointerException	1	3.5 / 1000	0.05	fr.irisa.triskell.eclipse.util
fr.irisa.triskell.eclipse.ecore	2	11.8 / 1000	0.13	fr.irisa.triskell.eclipse.util
EcoreHelper.java	2	14.2 / 1000	0.22	fr.irisa.triskell.eclipse.util
VariableNamingConventions	2	14.2 / 1000	0.22	fr.irisa.triskell.eclipse.util
fr.irisa.triskell.eclipse.wizard	2	3.5 / 1000	0.05	fr.irisa.triskell.eclipse.util
WizardMessages.java	1	83.3 / 1000	0.50	fr.irisa.triskell.eclipse.util
ClassWithOnlyPrivateConstructorsShouldBeFinal	1	83.3 / 1000	0.50	fr.irisa.triskell.eclipse.util
AbstractExampleWizard.java	1	9.3 / 1000	0.20	fr.irisa.triskell.eclipse.util
EmptyMethodInAbstractClassShouldBeAbstract	1	9.3 / 1000	0.20	fr.irisa.triskell.eclipse.util
fr.irisa.triskell.string	2	N/A	N/A	fr.irisa.triskell.eclipse.util
org.kermeta.utils.argumentsreader	10	N/A	N/A	fr.irisa.triskell.eclipse.util
org.kermeta.ecore.model.helper	5	N/A	N/A	fr.irisa.triskell.eclipse.util
org.kermeta.jface.preference	2	N/A	N/A	fr.irisa.triskell.eclipse.util

Outils

- CodePro AnalytiX



Outil de recherche DECOR



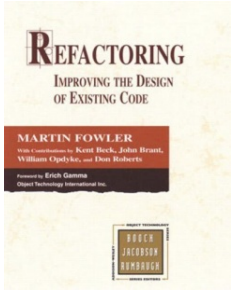
Fichier résultat de la détection

```
1.Name = SpaghettiCode
1.Class = org.apache.xerces.xinclude.XIncludeHandler
1.NoInheritance.DIT-0 = 1.0
1.LongMethod.MethodName = handleIncludeElement(XMLAttributes)
1.LongMethod.LOC_METHOD = 759.0
1.LongMethod.LOC_METHOD_Max = 254.5
1.GlobalVariable-0 = SYMBOL_TABLE
1.GlobalVariable-1 = ERROR_REPORTER
1.GlobalVariable-2 = ENTITY_RESOLVER
1.GlobalVariable-3 = BUFFER_SIZE
1.GlobalVariable-4 = PARSER_SETTINGS
...
2.Name = SpaghettiCode
2.Class = org.apache.xerces.impl.xpath.regex.RegularExpression
2.NoInheritance.DIT-0 = 1.0
2.LongMethod.MethodName = matchCharArray(Context,Op,int,int,int)
2.LongMethod.LOC_METHOD = 1246.0
2.LongMethod.LOC_METHOD_Max = 254.5
2.GlobalVariable-0 = WT_OTHER
2.GlobalVariable-1 = WT_IGNORE
2.GlobalVariable-2 = EXTENDED_COMMENT
2.GlobalVariable-3 = CARRIAGE_RETURN
2.GlobalVariable-4 = IGNORE_CASE
...
```

Outils

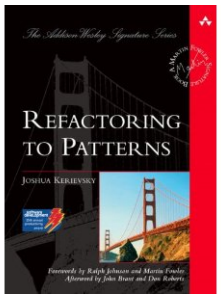
- Campwood Software
  - Calcul de métriques et complexité des modules
- Duploc
  - Outil pour la duplication de code
- CodePro AnalytiX
  - Assez similaire à Metrics

Refactorisation



- Définition
  - Une refactorisation (en anglais, refactoring) est une technique utilisée pour améliorer la structure interne d'un système sans en modifier le comportement externe en effectuant des changements dans le code source [Fowler, 1999].

- Refactorisation vers des patrons
  - Utiliser les patrons pour améliorer une conception existante



Exemple de refactorisation

- Encapsulate Field

```
public String _name;
```

Il y a un attribut public

Le rendre privé et fournir des accesseurs

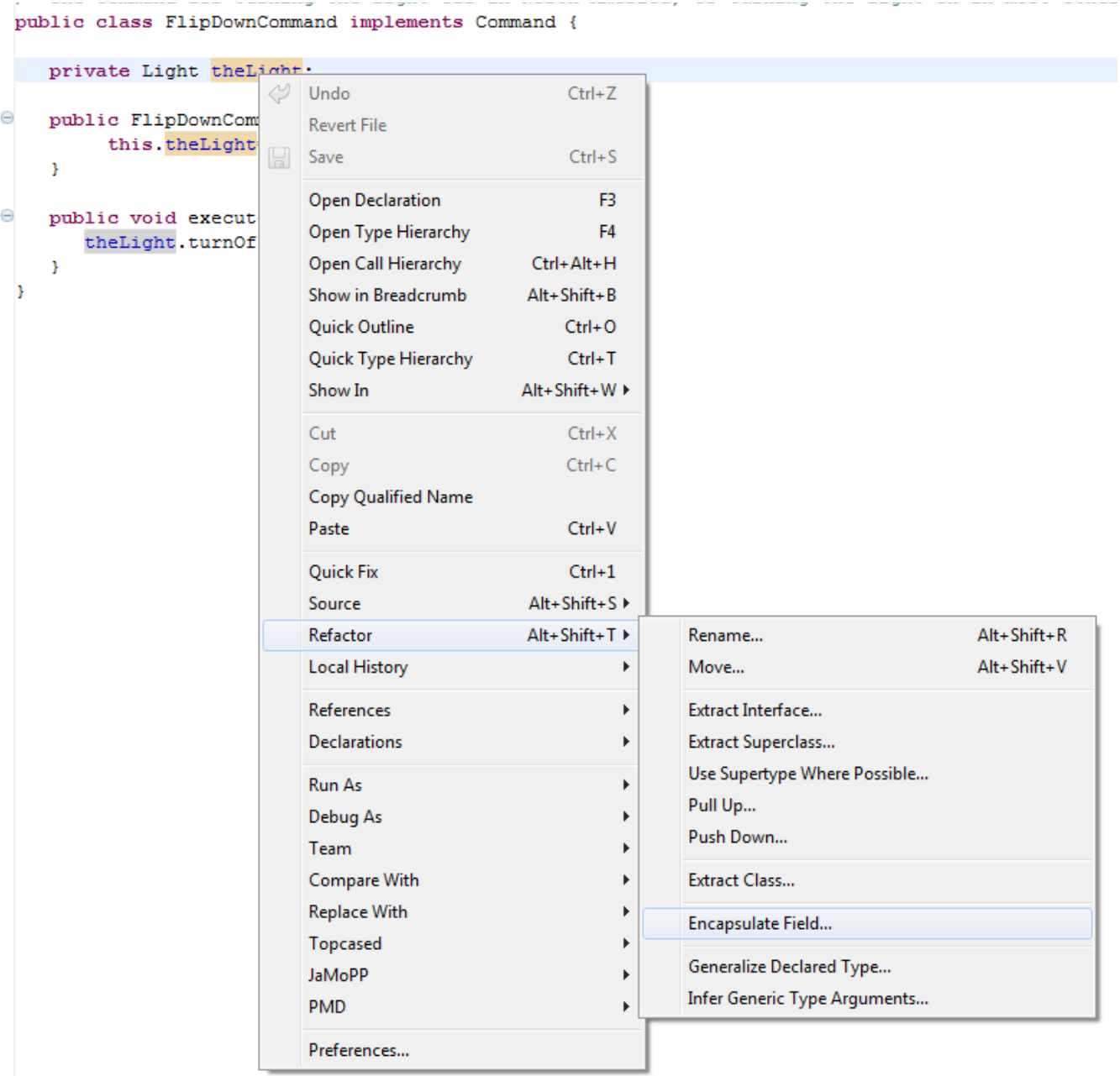
```
private String _name;
public String getName() {
  return _name;
}
public void setName(String arg) {
  _name = arg;
}
```

Exemple de refactorisation

- Encapsulate Field
  - Principe de l'encapsulation afin de permettre d'augmenter la modularité, faciliter la réutilisation et la maintenance du code
  - La représentation interne peut changer sans modifier l'interface externe

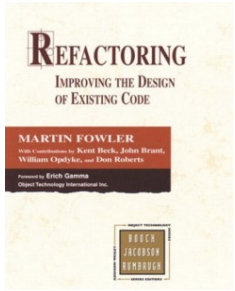
Outil de refactorisation

- Eclipse



Autres exemples de refactorisation

- Liste complète
  - http://refactoring.com/catalog/index.html
  - Refactoring: improving the design of existing programs. Martin Fowler. Addison-Wesley, 1999.

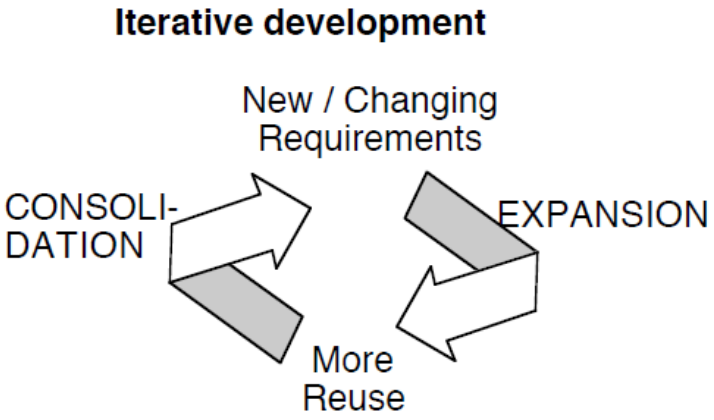


Pourquoi refactoriser ?

- Améliorer la conception d'un logiciel
- Réduire
  - le déclin et le vieillissement du logiciel
  - la complexité logicielle
  - les coûts de maintenance du logiciel
- Augmenter
  - la compréhension logicielle
    - En introduisant des patrons de conception
  - la productivité du logiciel (à long et court terme)
- Faciliter les futurs changements

Quand refactoriser ?

- Deux phases essentielles dans une approche itérative de développement logiciel
  - (Selon Foote et Opdyke, 1995)
  - Expansion
    - Ajouter de nouvelles fonctionnalités
  - Consolidation
    - Réorganiser et restructurer le logiciel
    - Introduire des patrons de conception
    - Appliquer des refactorisations
  - S'adapte naturellement dans un modèle de processus en spiral



Consolidation is necessary to ensure next expansion success

Quand refactoriser ?

- S’adapte également naturellement à la philosophie des méthodes agiles
  - Principes du manifeste agile
    - Une attention continue à l'excellence technique et à la qualité de la conception améliore l'agilité
    - La simplicité - l'art de maximiser la quantité de travail à ne pas faire - est essentielle

Quand refactoriser ?

- Dès que nécessaire
  - Pas à une période donnée
- La règle des 3
  - 1ère fois: vous devez implémenter quelque chose, faîtes-le!
  - 2ème fois: vous implémentez qqchose similaire en dupliquant du code
  - 3ème fois: ne pas ré-implémenter ou dupliquer, mais refactoriser!
- Lors de l'ajout d'une fonctionnalité (Consolidation)
  - Surtout quand la fonctionnalité est difficile à intégrer
- Lors du déboggage
  - Difficile de tracer une erreur, refactoriser pour rendre le code plus compréhensible
- Lors des revues de code

Quoi refactoriser ?

Défauts de conception	Refatorisations suggérées
Data Class	Move Method Encapsulate Field Encapsulate Collection
Duplicated Code	Extract Method Extract Class Form Template Method Pull Up Method Pull Up Field
Feature Envy	Extract Method Move Method Move Field
Large Class	Extract Class Extract Subclass Extract Interface
Switch Statement	Move Accumulation to Visitor Replace Conditional Dispatcher with Command Replace Conditional with Polymorphism Replace Type Code with Subclasses Replace Type Code with State/Strategy Replace Parameter with Explicit Methods

Références

- http://www.industriallogic.com/
- http://www.refactoring.com
- Refactoring: improving the design of existing programs. Martin Fowler. Addison-Wesley, 1999.
- AntiPatterns: Refactoring Software, Architectures and Projects in Crisis. Brown, et al. John Wiley & Sons, 1997.
- An introduction to Software Refactoring. Tom Mens, University of Mons-Hainaut.
- Software Evolution. Tom Mens, University of Mons-Hainaut.