



Génie Logiciel : Conception “Code as a Crime Scene”

UQÀM | Département d'informatique

Crédit Images: Pixabay & Pexels

Sébastien Mosser
INF 5153 - Cours #13 - A19



Récapitulatif livraisons clientes

	1	2	3	4	5	6	7	8	9	10	11	F
ISD	Red		Red									
ISE												
ISG												
ISJ	Red											
ISL	Red		Green	Red	Green	Green	Red	Green	Red			
ISO	Red	Red		Red			Red	Red	Green			
ISR		Green	Red	Red	Red	Red	Red	Red	Red			
ISS		Red		Green	Green	Red	Red	Red	Red			
IST			Green	Red	Green	Red	Red	White	Red			
ISU	Red	Red	Green	Red	Red	Red	Red	White	Red			
ISW		Green	Red	Red	Green	Green	Red	Red	Green			
ISZ	Red		Red	Red	Red	Red	Red	Red	Red			

The Pragmatic Programmers

Investigator:
Date:
Case #:
Location:

Your Code as a Crime Scene

Use Forensic Techniques to Arrest Defects, Bottlenecks, and Bad Design in Your Programs

Adam Tornhill

decodeMessage()

Tudor Gîrba

Michele Lanza

Radu Marinescu

Pragmatic Design Quality Assessment

https://fr.slideshare.net/girba/pragmatic-quality-assessment-tutorial-icse-2008

Pragmatic Design Quality Assessment

Tudor Gîrba

University of Bern, Switzerland

Michele Lanza

University of Lugano, Switzerland

Radu Marinescu

Politehnica University of Timisoara, Romania

[PDQA]

Les **problèmes de conception** sont ...

Fréquents

Inévitables

Dispendieux

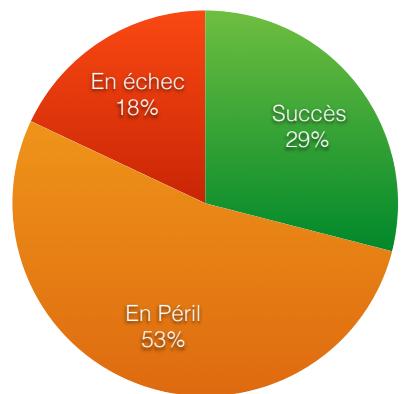
[PDQA]

Comment les éviter ?

On peut pas. Mais on peut
vivre avec et s'améliorer !

[PDQA]

Le logiciel, cet objet complexe



[The Stanford Group, 2004]

[PDQA]

1,000,000,000 de lignes de code

$\times 2s = 2,000,000,000$ secondes

$/ 3600 = 560$ heures

$/ 8 = 70$ jours

$/ 20 = 3$ mois !!

[PDQA]

Heureusement, on ne lit pas le code entièrement !

On lit les modèles de conception !

[PDQA]

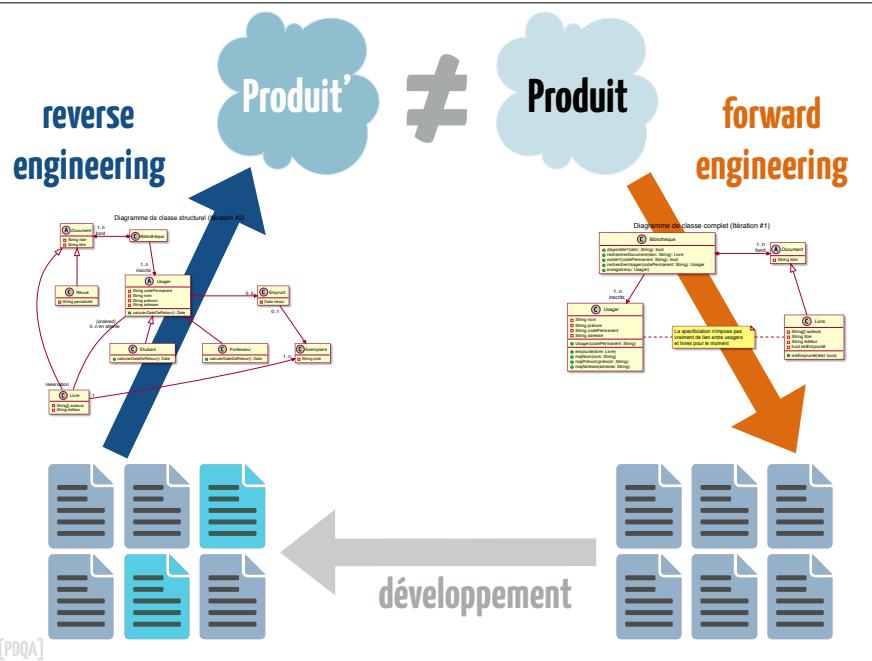
How the UML diagram describes the software



How the code is actually written



<https://twitter.com/JobOpportunitiesT/status/1106631344233295872?s=19>

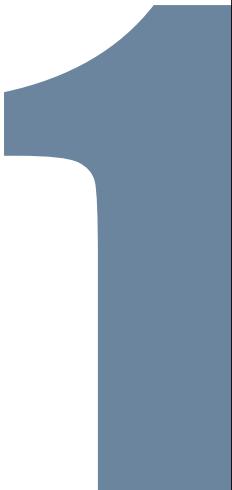


Un vrai système contient beaucoup de détails !

Comment juger de sa qualité ?

[PDQA]

Quantifier



You cannot **control**
what you cannot **measure**

Tom de Marco

[PDQA]

Métrique (définition)

Une **métrique** est une **fonction** qui assigne un nombre à un **produit**, un **processus** ou une **ressource**.

Métrique Logicielle (définition)

Les **métriques logicielles** sont des **mesures** associées à des **systèmes logiciels**, leurs **processus de développement** et les **documents associés**.

[PDQA]

[PDQA]

Exemples de métriques Orientée-Objet

Métrique	Définition
NOM	Nombre de Méthodes
NOA	Nombres d'Attributs
LOC	Nombre de Lignes de Code
NOS	Nombre d'instructions
NOC	Nombre de classes enfant

[PDQA]

Complexité Cyclomatique (CYCLO)

La **complexité cyclomatique** (ou nombre de McCabe) compte le **nombre de chemins indépendants à travers le code** d'une fonction

Indication sur le
nombre de tests à écrire

A part ça ...
pas grand chose

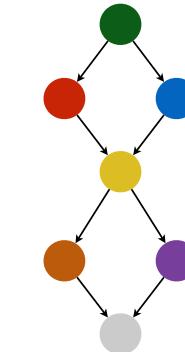
[PDQA]

CYCLO(P) ?

P =

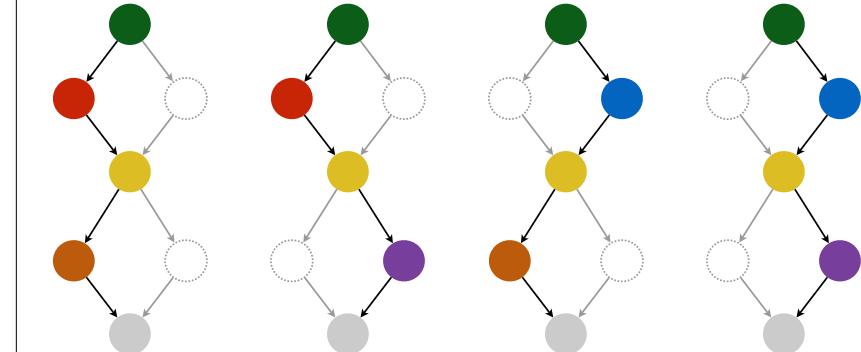
```
if( c1() )
    f1();
else
    f2();

if( c2() )
    f3();
else
    f4();
```



https://en.wikipedia.org/wiki/Cyclomatic_complexity

CYCLO(P) = 4



https://en.wikipedia.org/wiki/Cyclomatic_complexity

Méthodes pondérées (WMC)

“Weighted Method Count” (WMC) fait la somme pondérée des méthodes d'une classe (classiquement avec CYCLO pour pondérer)

Configurable, on peut changer la pondération au besoin

A part ça ...
pas grand chose

[PDQA]

Couplage entre objets (CBO)

Mesure du couplage en identifiant les méthodes et attributs utilisés depuis l'extérieur de la classe.

Prend en compte les vrais dépendances

Pas de mesure de l'intensité du couplage

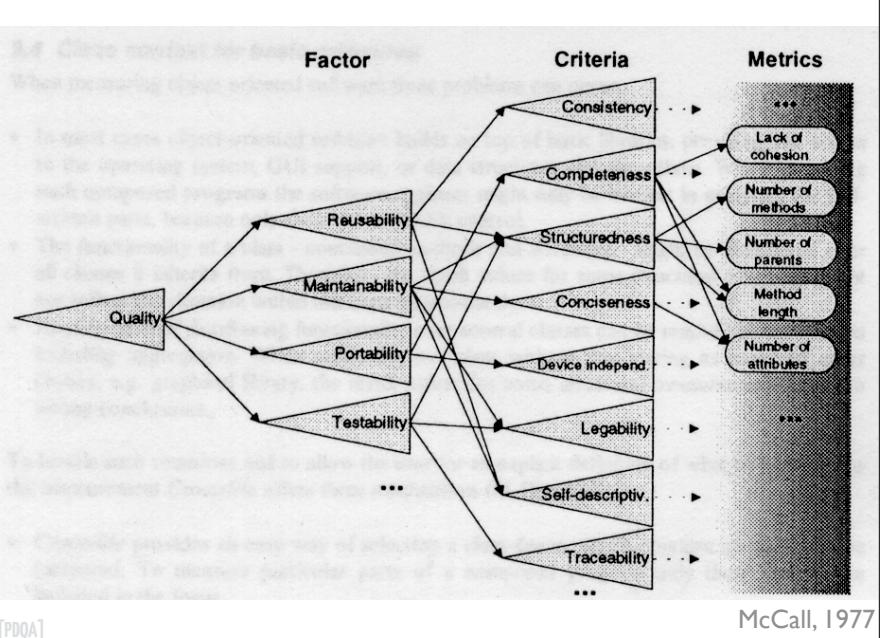
Profondeur d'héritage (DIT)

“Depth of Inheritance Tree” (DIT) est la profondeur maximale de l'arbre d'héritage pour une classe donnée.

Donne une quantification à la notion d'héritage

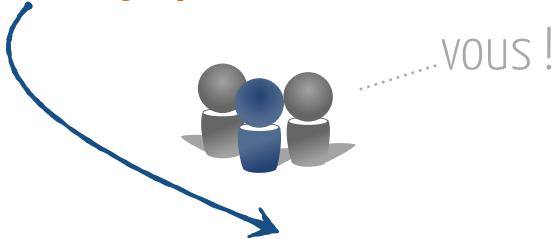
Comment l'interpréter ?

[PDQA]



Le problème #1 avec les métriques ...

Capture un symptôme



Comment en tirer le traitement ?

[PDQA]

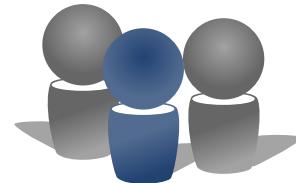
Le problème #2 avec les métriques ...

SOLID

GRASP

DRY

KISS



On raisonne sur des **principes**, pas des **métriques**

[PDQA]

Exploiter 2

Metric	Value
LOC	35175
NOM	3618
NOC	384
CYCLO	5579
NOP	19
CALLS	15128
FANOUT	8590
AHH	0.12
ANDC	0.31

En quoi ça nous aide ?

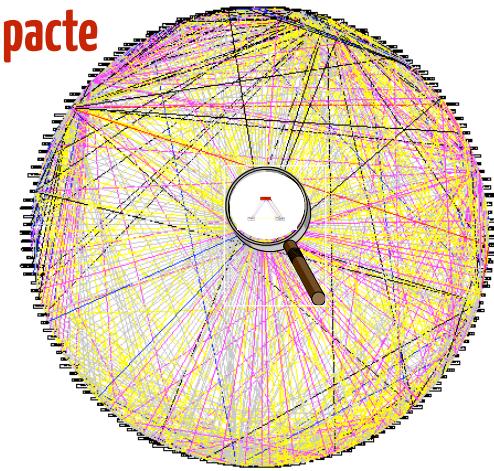
[PDQA]

En quoi ça nous aide ?

Si je change
juste cette méthode ...



Mais qu'en fait ça impacte
33% du code



C'est l'analyse conjointe des métriques
qui aide les développeurs.

Votre Premier Anti-Patron

Un **BLOB** (ou **classe Dieu**) tend à **centraliser**
toute l'intelligence du système, à **tout faire** et à **utiliser** des données en
provenance de **classes purement**
structurelle



Un **BLOB** :



- **centralise toute l'intelligence** du système;
- **fait tout**;
- **utilise** des données de **classes structurelle**.

Un BLOB :



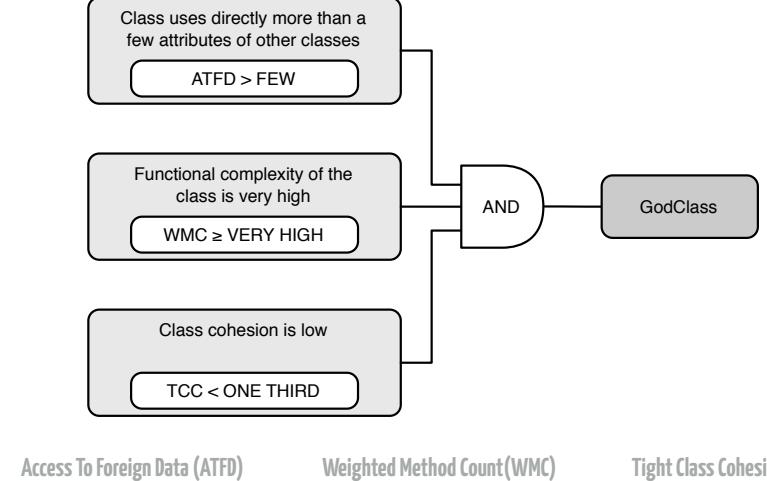
- Est complexe
- N'est pas cohésif
- Utilise des données externes

[PDQA]

A God Class centralizes too much intelligence in the system.

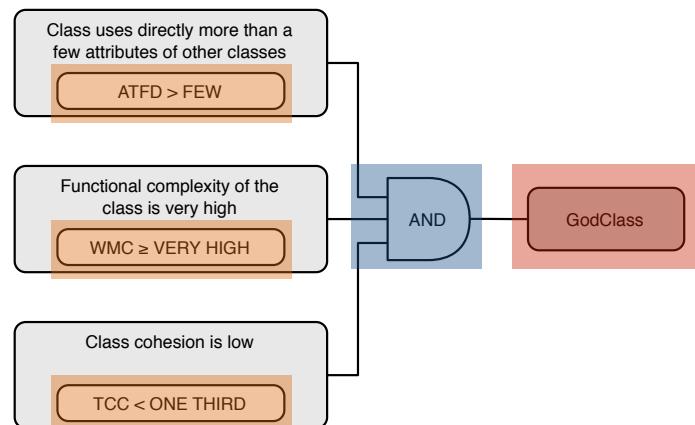
Lanza, Marinescu 2006

[PDQA]



A God Class centralizes too much intelligence in the system.

Lanza, Marinescu 2006



Composition de métrique

Problème de conception

[PDQA]

Class uses directly more than a few attributes of other classes
ATFD > FEW

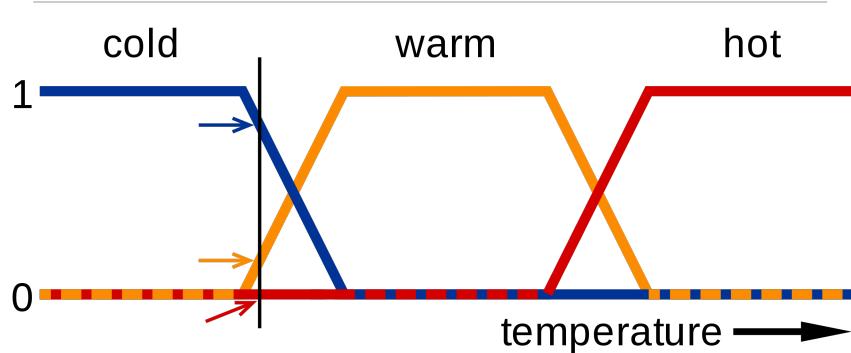
Functional complexity of the class is very high
WMC ≥ VERY HIGH

Class cohesion is low
TCC < ONE THIRD

Comment
Définir
Les
Seuils ?

[PDQA]

Définition des seuils



Principes de logique dite “floue”

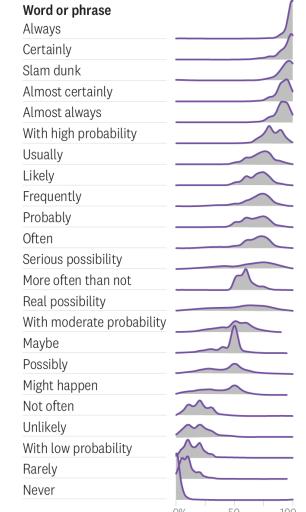
https://en.wikipedia.org/wiki/Fuzzy_logic

Il est toujours difficile de quantifier les seuils de detection

Qui pour les définir ?

How People Interpret Probabilistic Words
“Always” doesn't always mean always.

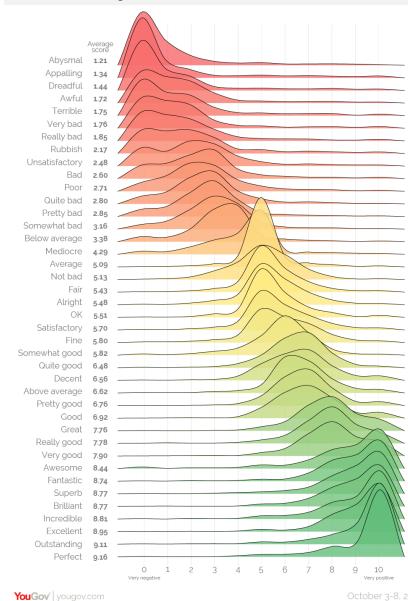
Distribution of responses according to respondents' estimate of likelihood



<https://hbr.org/2018/07/if-you-say-something-is-likely-how-likely-do-people-think-it-is>

Source: Andrew Mauboussin and Michael J. Mauboussin © HBR

How good is “good”? Now with even more words!
On a scale of 0 to 10, where 0 is ‘very negative’ and 10 is ‘very positive’, in general, how positive or negative would the following word/phrase be to someone when you used it to describe something?



An Envious Method is more interested in data from a handful of classes.

Lanza, Marinescu 2006

Method uses directly more than a few attributes of other classes
 $ATFD > FEW$

Method uses far more attributes of other classes than its own
 $LAA < ONE\THIRD$

The used “foreign” attributes belong to very few other classes
 $FDP \leq FEW$

AND

Feature Envy

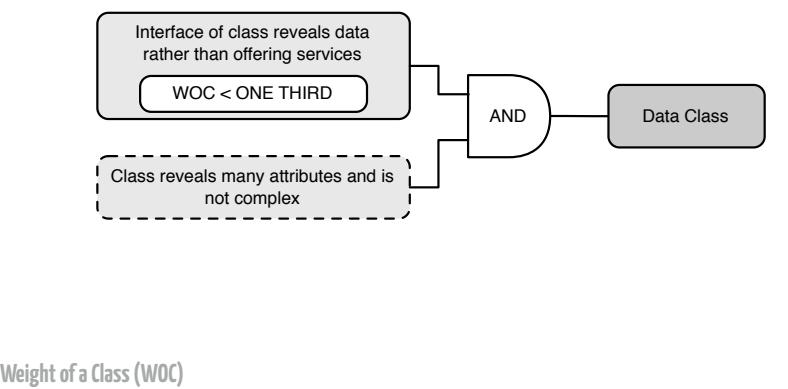
Locality of Attribute Access (LAA)

Foreign Data Providers (FDP)

[PDQA]

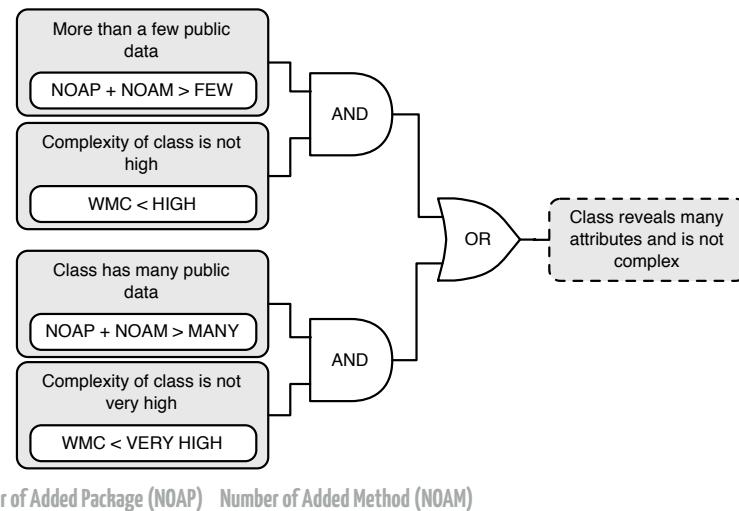
Data Classes are dumb data holders.

Lanza, Marinescu 2006

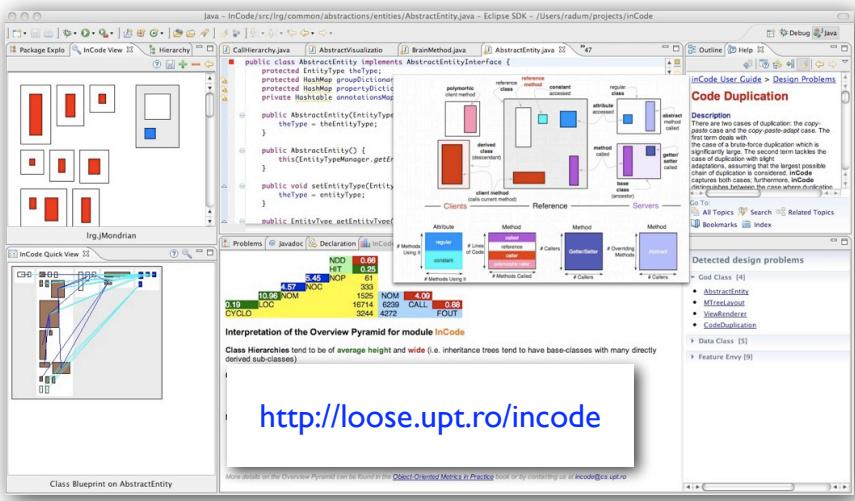


Data Classes are dumb data holders.

Lanza, Marinescu 2006



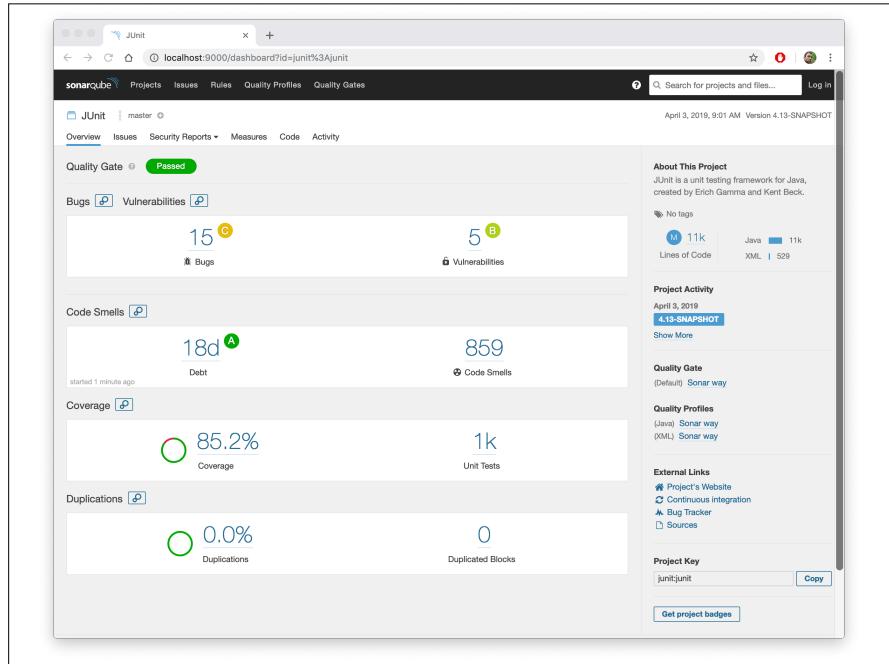
L'analyse de qualité fait partie du cycle de vie



Exemple Industriel : SonarQube

```
lucifer:tmp$ git clone https://github.com/junit-team/junit4.git
lucifer:tmp$ cd junit4
lucifer:junit4$ mvn org.jacoco:jacoco-maven-plugin:prepare-agent \
               clean verify
lucifer:junit4$ mvn sonar:sonar
```

Fonctionne avec n'importe quel projet Java géré par Maven



Visualiser

3



Peut-on mesurer la beauté d'une oeuvre en comptant le nombre de couleurs utilisées ?

[PDQA]

Les métriques seules sont peu utile

Metric	Value	Remarks
No. of Lines of Code	223,068	including comments
No. of Source Files	1,209	*.java files
No. of Packages	99	-
No. of Classes	1,393	including 140 inner classes
No. of Methods	9,561	including accessor methods
No. of Attributes	3,358	all variables including static and local variables

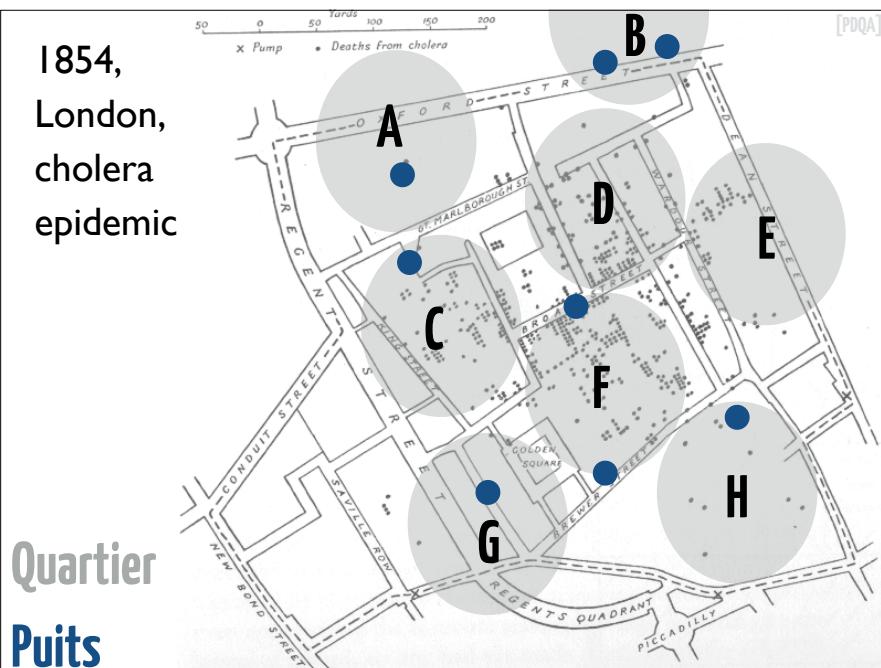
On peut les combiner,
mais le résultat reste binaire

[PDQA]

Épidémie de Choléra à Londres (1854)

Quartier	Puits	Malade
A	1	Aucun
B	2	Aucun
C	1	Élevé
D	0	Élevé
E	0	Faible
F	2	Élevé
G	1	Faible
H	1	Faible

1854,
London,
cholera
epidemic



Quartier
Puits

1854,
London,
cholera
epidemic

Foyer
Infectieux

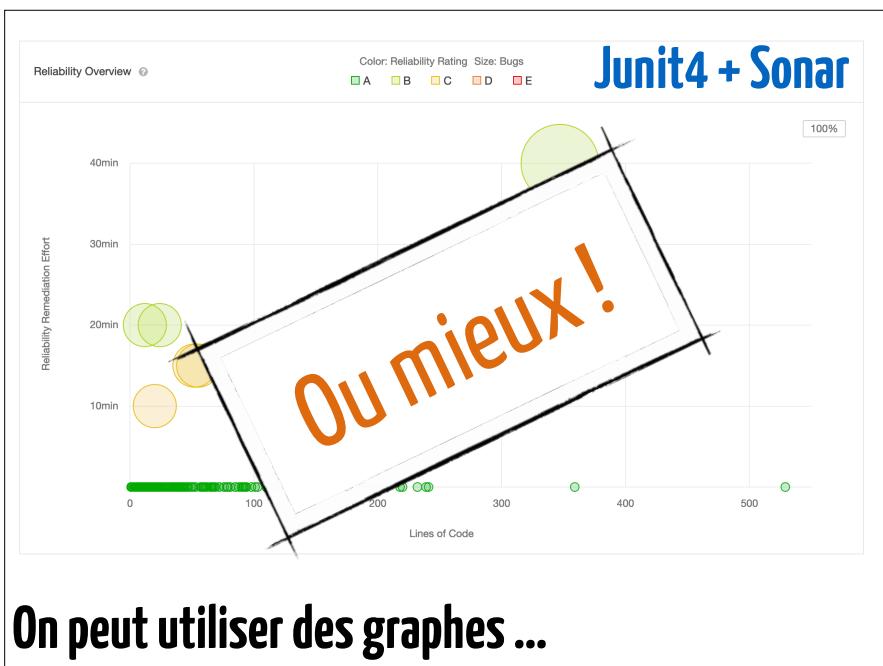
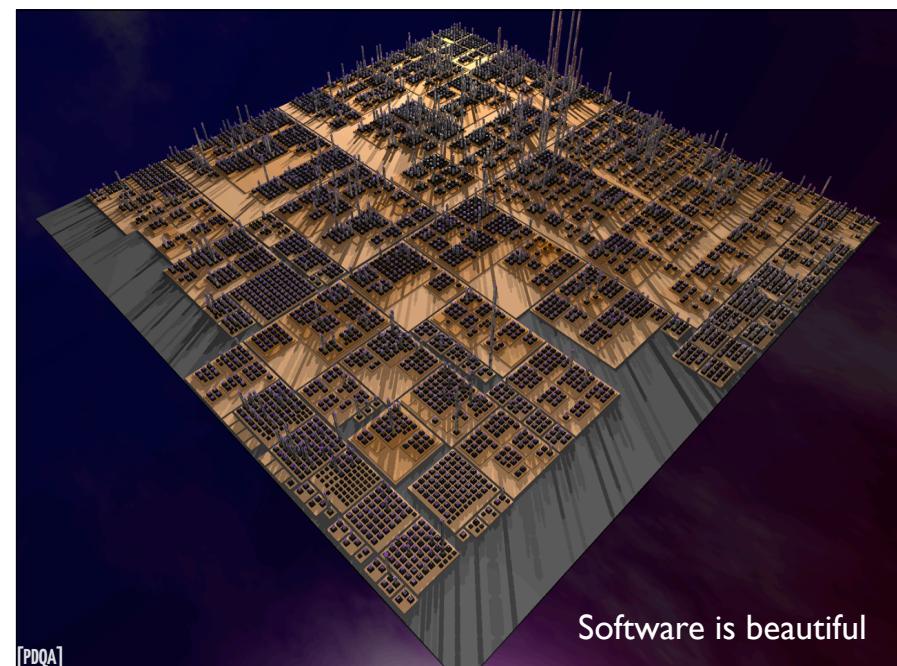
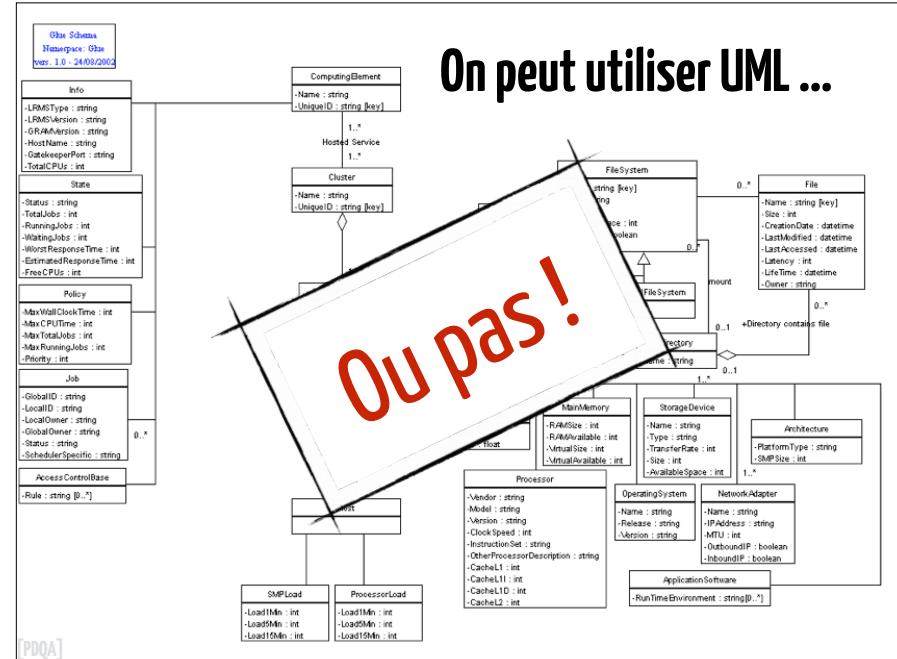
Puits



Épidémie de Choléra à Londres (1854)

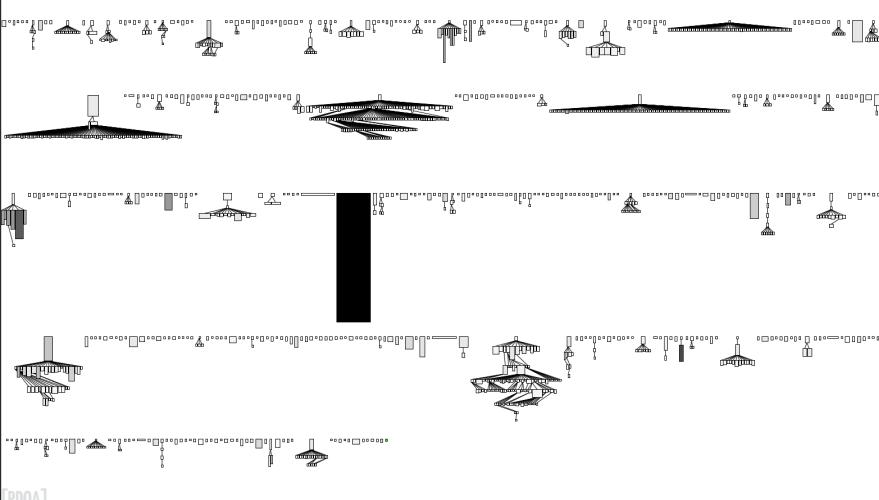
Quartier	Puits	Malade
A	1	Aucun
B	2	Aucun
C	1	Élevé
D	3	Élevé
E	0	Faible
F	2	Élevé
G	1	Faible
H	1	Faible

La bonne visualisation est la distance au puit

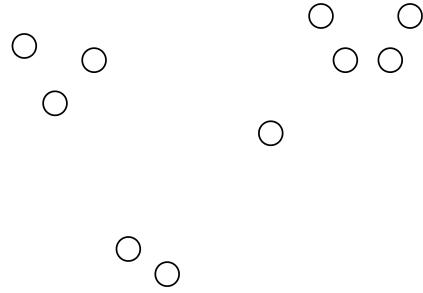


On peut utiliser des graphes ...

La visualisation rend la conception visible



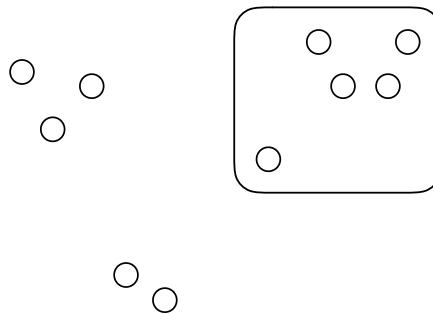
Combien de groupes voyez vous ?



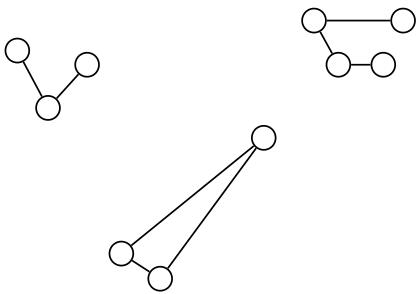
L'être humain est biologiquement capable de reconnaître des motifs.



Combien de groupes voyez vous ?

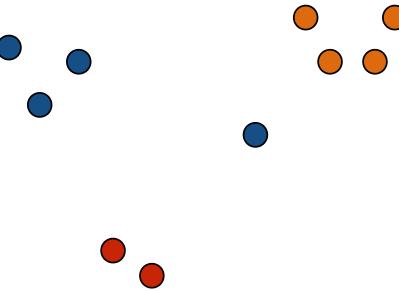


Combien de groupes voyez vous ?



[PDQA]

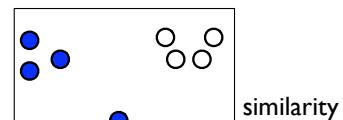
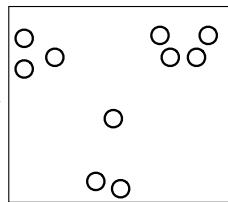
Combien de groupes voyez vous ?



[PDQA]

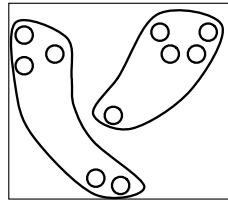
Principe de Gestalt

proximity



similarity

enclosure



connectivity

Visualiser = Exploiter la Pré-attention

8789364082376403128764532984732984732094873290845
389274-0329874-32874-23198475098340983409832409832
049823-0984903281453209481-0839393947896587436598

[PDQA]



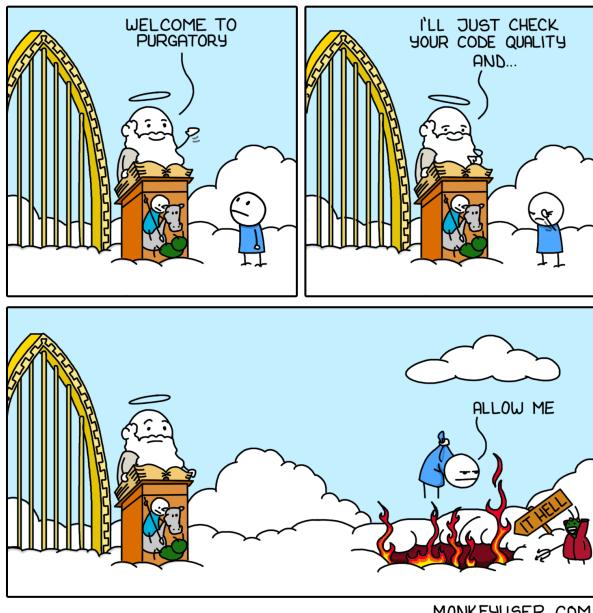
<http://blog.dinett-illustration.com/>



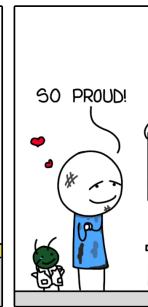
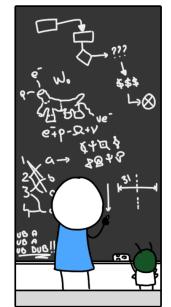
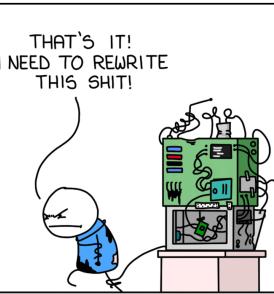
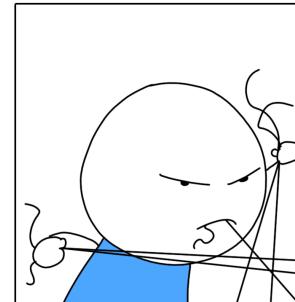
Analyser

4

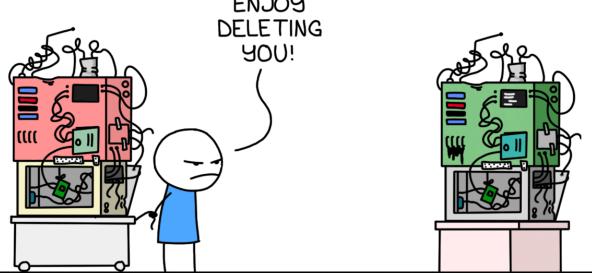
LAST PUSH



THAT'S IT!
I NEED TO REWRITE
THIS SHIT!



I'M GOING TO
ENJOY
DELETING
YOU!



Maintenir la qualité, c'est compliqué

Qu'est-ce qu'un
anti-patron ?

Votre Premier Anti-Patron

Un **BLOB** (ou **classe Dieu**) tend à **centraliser toute l'intelligence** du système, à **tout faire** et à **utiliser** des données en provenance de **classes purement structurelle**



Le poltergeist

Classes inutiles
(sans responsabilité)



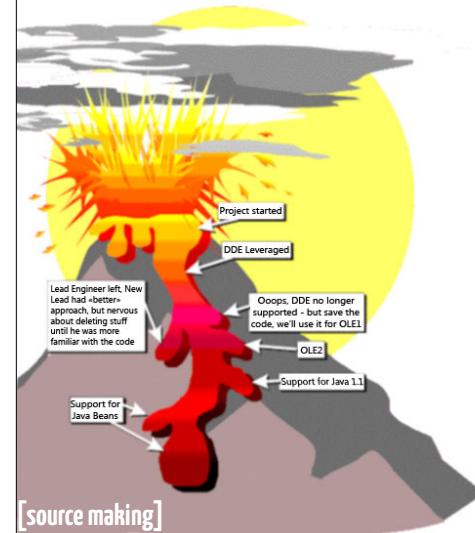
[source making]

Cycle de vie
(ultra court)

Le fleuve de lave

Mauvaises décisions prises

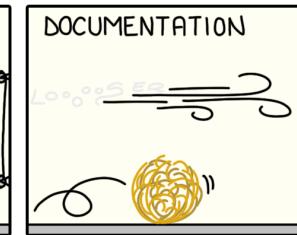
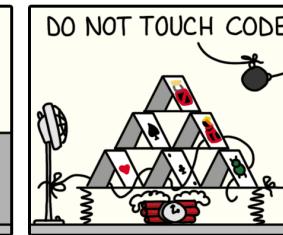
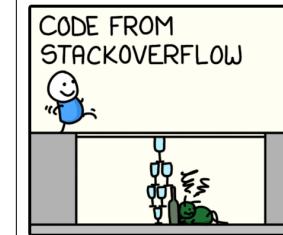
Code “mort” dans la base de code



Le champ de mine

Bug mineur mais **conséquences majeures**

Absence de maîtrise des objets conçus



Et tous les autres

- Le code spaghetti / la programmation copier-coller
 - Fort couplage, tout est entrelacé, cauchemar de maintenance
- Le Marteau doré (Golden Hammer)
 - Un outil / une technologie / une méthodologie pour les gouverner tous.
 - Aussi appelé : "MVC everywhere", "RoR everywhere", ...
- La balle d'argent (Silver Bullet)
 - Recherche de la solution ultime qui résoudra TOUS les problèmes
 - Même ceux qu'on connaît pas encore

La classe Couteau-Suisse

Ajout de **fonctionnalités non cohésives**



Manque de responsabilisation des entités

Bikeshedding

Passer beaucoup de temps à **discuter de détails inutiles** (le garage à vélo)

Ne plus avoir le temps de se concentrer
SUR ce qui est important (la centrale nucléaire)

"Signal d'alarme"-driven development



Attendre jusqu'à la **catastrophe** pour faire plier les **exigences de qualité**

Permet de **rogner sur les tests**, la **documentation**, ...

"Si vous attendez la dernière minute, ça ne prend qu'une minute à faire" (Loi de Parkinson)

Améliorer

5

“To refactor”

To restructure software by applying a series of refactorings without changing its observable behavior

“Refactoring”

A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior

Quelle rapport avec la conception ?

- Le réusinage est un complément à la conception amont
 - Rappel : la conception parfaite n'existe pas, on cherche à tendre vers la moins pire
- Le travail d'analyse / conception amont est un travail d'anticipation
- Le réusinage permet de réparer ce qui n'as pas été vu lors de la conception
 - Mais entre temps on a livré de la valeur
- Principe de l'architecture émergente
 - Requiert de l'expérience, et n'est pas adapté à toutes les équipes.

Pourquoi ré-usiner le code ?

- Les programmes ...
 - ... complexe à lire sont complexes à modifier;
 - ... avec de la duplication dans la logique d'affaire sont complexes à modifier;
 - ... avec une logique d'affaire pleines de conditions sont complexes à modifier;
- ... sont complexes à modifier !
- On définit le concept de "dette technique" (technical debt)
 - Il est normal de s'endetter au démarrage du développement d'un produit
 - Mais si on ne rembourse pas sa dette, on fait faillite (anti-patrons, bad smells)

Role des Tests

A change made to the internal structure of software to make it easier to understand and cheaper to modify **without changing its observable behavior**

Quand ré-usiner ? (règle de 3)

The first time you do something, you **just do it.**

The second time you do something similar, you **wince at the duplication**, but you **do the duplicate** thing anyway.

The third time you do something similar, **you refactor**

Concrètement, c'est quoi un ré-usinage ?

- Peut agir sur le code, à plusieurs niveau
 - Méthode, classe
 - Structure (p.-ex., déplacer des méthodes entre objets)
 - Logique d'affaire
- C'est une transformation iso-fonctionnelle du code
 - Le comportement reste le même
 - On améliore la structure pour la clarifier, et améliorer la conception
- Ne pas mélanger ajout fonctionnel et ré-usinage

Exemples de ré-usinage

- Extract method:
 - Fragment de code qui peut être sémantiquement regroupé ensemble
 - Transformer ce fragment en une méthode qui porte le nom associé à la sémantique
- Inline method:
 - Méthode dont le corps est tout aussi parlant que son appel
 - Remplacer les appels à la méthode par le corps de celle-ci

Comment prendre la décision ?

Les problèmes avec le ré-usinage

- Consomme du temps sur le temps de développement
- Peut avoir un impact sur les performances (à vérifier avec un banc de test)
- Est facilité lorsque l'outilage est adapté
 - Les IDE classiques proposent des outils de ré-usinage
 - Apprenez à les utiliser, c'est parfois "surprenant"
- Lien avec les architectures en couches :
 - On peut ré-usiner comme on veut tant qu'on ne touche pas aux interfaces publiques.

Améliorer du logiciel

- On peut faire du réusinage à la main
 - Cela demande de l'expertise et du temps
- On peut aussi automatiser ces opérations
 - "Pourquoi écrire des programmes quand on peut écrire des programmes qui les écrivent à notre place ?"
- Exemples :
 - Académique : Repairnator, un bot GitHub qui fixe des bugs et fabrique les PR
 - Industriel : coccinelle, outil de réparation du code source de Linux

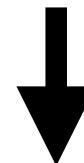
Contexte : Réécriture du noyau Linux



857,452
“commits”

03.09.2019, 09:57 (CEST)

```
...  
x = kmalloc(sizeof(*a), 0);  
memset(a, 0, sizeof(*a))  
...
```



```
...  
x = kzalloc(sizeof(*a), 0);  
...
```

Coccinelle (Lawall, Muller et al)

Principe de “patch” sémantique (Coccinelle)

```

1 @@
2 type T;
3 expression x, E, E1, E2;
4 @@
5 - x = kmalloc(E1, E2);
6 + x = kzalloc(E1, E2);
7 ... when != \(\ x[...]=E; \ / \ x=E; \ )
8 - memset((T) x, 0, E1);

(a) kmalloc^memset(0) ↪ kzalloc (Rk)

```

Réécriture : Programme → Programme

93

Composition : Réutilisation des réécritures

```

1 @@
2 type T;
3 expression x, E, E1, E2;
4 @@
5 - x = kmalloc(E1, E2);
6 + x = kzalloc(E1, E2);
7 ... when != \(\ x[...]=E; \ / \ x=E; \ )
8 - memset((T) x, 0, E1);

```

```

1 struct Point {
2     double x;
3     double y;
4 };
5 typedef struct Point Point;
6
7 int main()
8 {
9     Point *a;
10    // ...
11    a = kmalloc(sizeof(*a), 0);
12    // not using a
13    memset(a, 0, sizeof(a));
14    // ...
15    return 0;
16 }

```

```

1 @@
2 type T;
3 T *x;
4 expression E;
5 @@
6
7 - memset(x, E, sizeof(x))
8 + memset(x, E, sizeof(*x))

```

Règles de Réécriture

Code source cible

94

Comp

Semantic patches and patches related to collateral evolutions
Patches that have been integrated into the official Linux tree (Linus Torvalds' repository):

- Use UPIO_MEM. [1](#) ([semantic_patch](#))
- Use resource_size. [1](#) [2](#) ([semantic_patch](#))
- Use dev_get_drvdata. [1](#) [2](#) ([semantic_patch](#))
- use usb_get_intdata, usb_set_intdata. [1](#) [2](#) ([semantic_patch](#))
- use DEFINE_SPINLOCK. [1](#) [2](#) [3](#) [4](#) [5](#) ([semantic_patch](#))
- use ARRAY_SIZE. [1](#) [2](#) [3](#) [4](#) ([semantic_patch](#))
- use DIV_ROUND_CLOSEST. [1](#) [2](#) [3](#) [4](#) ([semantic_patch](#)) [4](#) ([semantic_patch](#))
- use DIV_ROUND_UP. [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) ([semantic_patch](#)) [9](#) [10](#) [11](#) [12](#) ([semantic_patch](#))
- use BUG_ON. [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) ([semantic_patch](#))
- use FIELD_SIZEOF. [1](#) ([semantic_patch](#))
- use time_before, time_before_eq, etc. [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) ([semantic_patch](#))
- [potential-error-irqfunc.patch](#) ([semantic_patch](#))
- [kzalloc-treewide.patch](#) ([semantic_patch](#))

Bug fixing patches

- Use kzalloc for allocating only one thing [1](#) [2](#) ([semantic_match](#))
- Fix size given to memset [1](#) [2](#) [3](#) [4](#) [5](#) ([semantic_match](#))
- Size of a pointer [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) ([semantic_match](#))
- Use PTR_ERR to get error code [1](#) ([semantic_match](#))
- Use ERR_PTR/IS_ERR to return a flag as a pointer [1](#) ([semantic_match](#))
- Remove exceptional

tures

R2

P'

59! > nombre d'atomes
dans l'univers



95

Composition : Réutilisation des réécritures

```

1 @@
2 type T;
3 expression x, E, E1, E2;
4 @@
5 - x = kmalloc(E1, E2);
6 + x = kzalloc(E1, E2);
7 ... when != \(\ x[...]=E; \ / \ x=E; \ )
8 - memset((T) x, 0, E1);

```

```

1 struct Point {
2     double x;
3     double y;
4 };
5 typedef struct Point Point;
6
7 int main()
8 {
9     Point *a;
10    // ...
11    a = kmalloc(sizeof(*a), 0);
12    // not using a
13    memset(a, 0, sizeof(a));
14    // ...
15    return 0;
16 }

```

```

1 @@
2 type T;
3 T *x;
4 expression E;
5 @@
6
7 - memset(x, E, sizeof(x))
8 + memset(x, E, sizeof(*x))

```

R2(~~X~~(P)) = P''

R1

P

?

Pas d'erreurs
au sens de Coccinelle

96

1ère étape : Un modèle pour raisonner

Let $\rho = (\varphi, \chi) \in (\Phi \times X) = P$, $(\varphi : AST \rightarrow AST) \in \Phi$

$\chi : AST \times AST \rightarrow \mathbb{B} \in X$, $\forall p \in AST, \chi(p, \varphi(p))$

Indépendance Technologique

$apply : AST \times P_<^n \rightarrow AST$

$p, [\rho_1, \dots, \rho_n] \mapsto \text{Let } p_{2,n} = (\bullet_{i=2}^n \varphi_i)(p), p' = \varphi_1(p_{2,n}), \chi_1(p_{2,n}, p')$

Isofonctionnalité avec l'outilage existant

97

2nde étape : Déetecter les problèmes

$seq : AST \times P_<^n \rightarrow AST$

$p, [\rho_1, \dots, \rho_n] \mapsto p_{seq} = (\bullet_{i=1}^n \varphi_i)(p)$,

$\wedge_{i=1}^n \chi_i(p, p_{seq})$

R2(P'') = P'' 
R1(P'') ≠ P''

R2(P') = P' 
R1(P') = P'

98

Trouver l'ordre est un problème ... Et si on pouvait s'en passer ?

$iso : AST \times P^n \rightarrow AST$

$p, \{\rho_1, \dots, \rho_n\} \mapsto p_{iso} = p \oplus (\cdot_{i=1}^n (\varphi_i(p) \ominus p)), \quad \wedge_{i=1}^n \chi_i(p, p_{iso})$

$\oplus : AST \times A_<^* \rightarrow AST$

$(p, S) \mapsto \begin{cases} S = \emptyset & \Rightarrow p \\ S = \alpha | S' \Rightarrow exec(\alpha, p) \oplus S' \end{cases}$

$\ominus : AST \times AST \rightarrow A_<^*$

$(p', p) \mapsto \Delta$, where $p' = p \oplus \Delta$ Hypothèse : Opérateur de Diff

99

Example Java

RunnerUp



(<https://github.com/jonasoreland/runnerup>)

4 règles de réparation énergétiques



24 séquences différentes

Composition isolée 

Composition séquentielle 

Validation sur le noyau Linux

100

A Delta-oriented Approach to Support the Safe Reuse of Black-box Code Rewriters

Benjamin Benni^{*1} | Sébastien Mosser² | Naouel Moha² | Michel Riveill¹

¹Université Côte d'Azur, CNRS, I3S, Nice, France

²Université du Québec à Montréal, Montréal, Québec, Canada

Correspondence

*Benjamin Benni Email:
benetti@i3s.unice.fr

Summary

Large-scale corrective and perfective maintenance is often automated thanks to rewriting rules using tools such as Python2to3, Spoon or Coccinelle. Such tools consider these rules as black-boxes and compose multiple rules by chaining them: giving the output of a given rewriting rule as input to the next one. It is up to the developer to identify the right order (if it exists) among all the different rules to yield the right program. In this paper, we define a formal model compatible with the black-box assumption that reifies the modifications (Δ s) made by each rule. Leveraging these Δ s, we propose a way to safely compose multiple rules when applied to the same program by (i) ensuring the isolated application of the different rules and (ii) identifying unexpected behaviors that were silently ignored before. We assess this approach on two large-scale case studies: (i) identifying conflicts in the Linux source-code automated maintenance and (ii) fixing energy anti-patterns existing in Android applications available on GitHub.

KEYWORDS:

Code Rewriting, Software Reuse, Conflict detection, Rule Composition

<https://hal.archives-ouvertes.fr/hal-01722040?lang=en>

Et si il y a plusieurs
anti-patrons au
même endroit ?

```
1 public class C {  
2     private String data;  
3     public String setData(String s) {  
4         this.data = s;  
5     }  
6     public void doSomething() {  
7         ...  
8         setData(newValue) /* <<< */;  
9         ...  
10    }  
11 }  
12 // ...  
13 }  
14 }
```

```
1 public class C {  
2     private String data;  
3     public String setData(String s) {  
4         this.data = s;  
5     }  
6     public void doSomething() {  
7         ...  
8         this.data = newValue /* <<< */;  
9         ...  
10    }  
11 }  
12 // ...  
13 }  
14 }
```

(a) Example of a Java class (C.java, p_j)

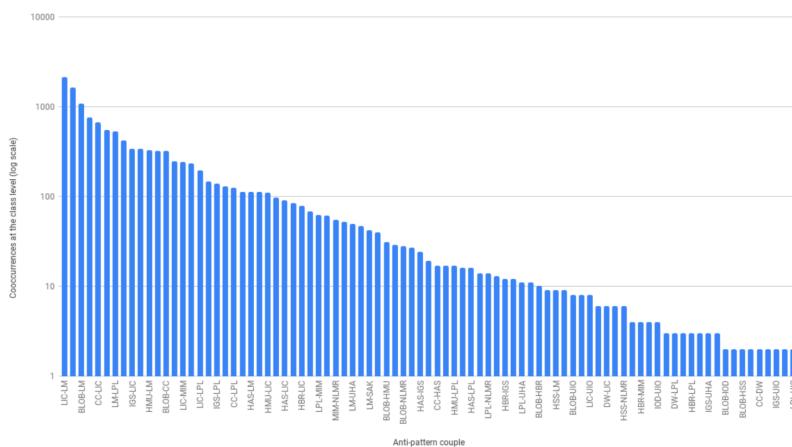
$$(b) p_{igs} = R_{igs}(p_j)$$

```
1 public class C {  
2     private String data;  
3     public String setData(String s) {  
4         if (s != null)  
5             this.data = s;  
6     }  
7     public void doSomething() {  
8         ...  
9         this.data = newValue /* <<< */;  
10    }  
11 }  
12 // ...  
13 }  
14 }
```

Réparation automatique par ré-écriture de code

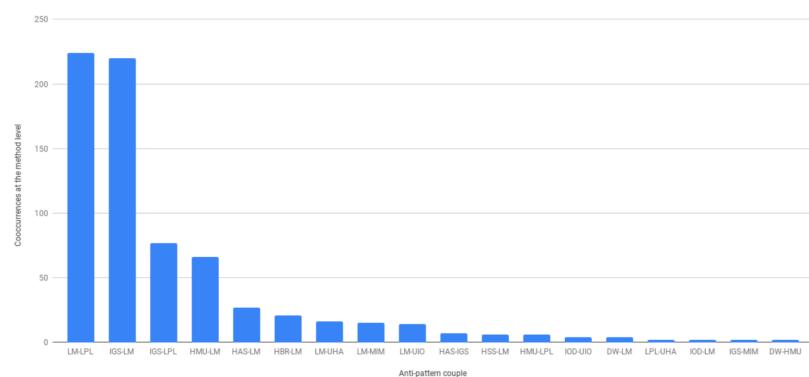
$$(d) p_{npoigs} = R_{np}(R_{igs}(p_j))$$

Co-located anti-patterns detected in Android apps (Class level)

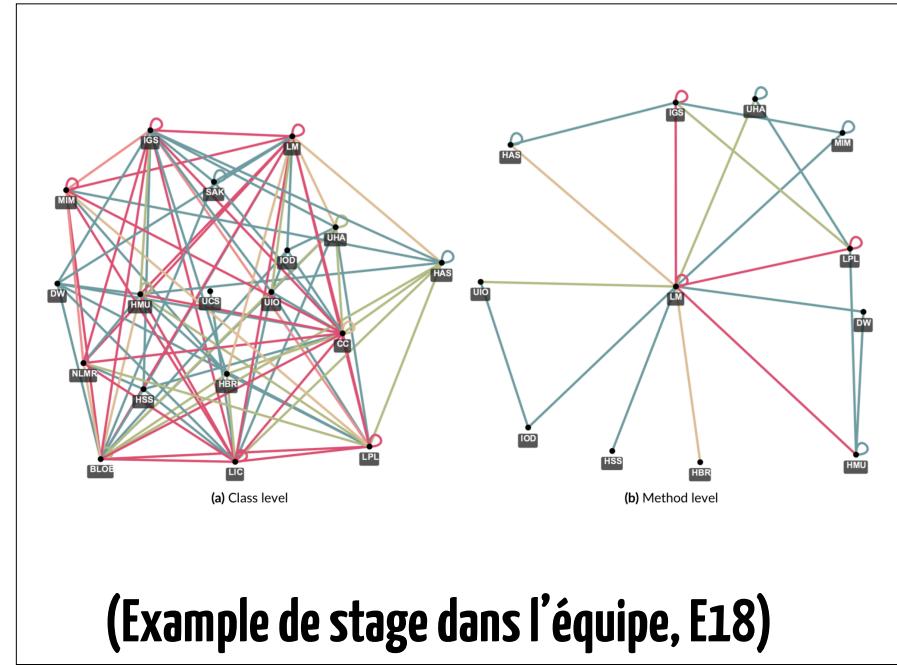


(a) Overlapping anti-patterns detected at the class level

Co-located anti-patterns detected in Android apps (Method level)



(b) Overlapping anti-patterns detected at the method level



(Example de stage dans l'équipe, E18)