Point sur l'organisation

Dernière semaine de PT & M331

le 10/12/2018, Mme Mireille Blay-Fornarino et Mme Nathalie Daumet-Fénéon

Organisation générale

1. Chaque équipe s'organise avec un P.O. (responsable de la valeur des réalisations) et un scrum Master (responsable de la méthode).
2. Chaque journée commence par un scrum meeting où tout le monde doit être présent.
    1. Tous les membres de l'équipe debout, SANS ORDINATEUR, présente brièvement
        1. où j'en suis, ce que j'ai fait hier
        2. ce que je vais faire aujourd'hui
        3. les problèmes rencontrés mais SANS les DÉCRIRE. A l'issue du Scrum meeting, le problème sera éventuellement discuté par l'équipe.
3. Chaque journée se termine par une démonstration où tout le monde doit être présent.
    1. Lors de cette démo, les US réalisées dans la journée sont présentées d'un point de vue "utilisateur" ; ce n'est pas les codes qui nous intéressent.
    2. A cette étape, il est possible de revoir les objectifs du lendemain.

Organisation dans la semaine

1. **LUNDI** : Chaque équipe organise les "histoires" et les tâches à réaliser jour par jour :
    a. exemple : Lundi histoire #1 et #2 seront livrées, les tâches X, Y ,Z seront terminées. Mardi histoire #3 ….. Cette répartition ("SPRINT" à la journée) est saisie dans les milestones correspondants (lundi, mardi, mercredi, jeudi).
    b. La répartition exige un travail important du P.O. et de l'équipe pour évaluer la complexité des tâches et sa "vélocité" pour atteindre ses objectifs.
2. **JEUDI** : Attention à bien prévoir de travailler la soutenance, la démo etc.

Evaluation

Cette semaine donnera lieu à plusieurs évaluations :

- "Organisation in situ" par Mme Fénéon.
- Qualité de la gestion des projets visible au travers des outils par Mme Blay.
- A l'examen, des questions porteront sur la Gestion de Projet de cette semaine.

Présence des enseignantes

Nous ne pourrons pas être présentes tout le temps.
Ne tenez pas compte des affectations aux groupes, nous tournerons sur les groupes et donc les salles oh oh oh...

Qualité du logiciel ?

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/

Qualité du logiciel ?

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?*

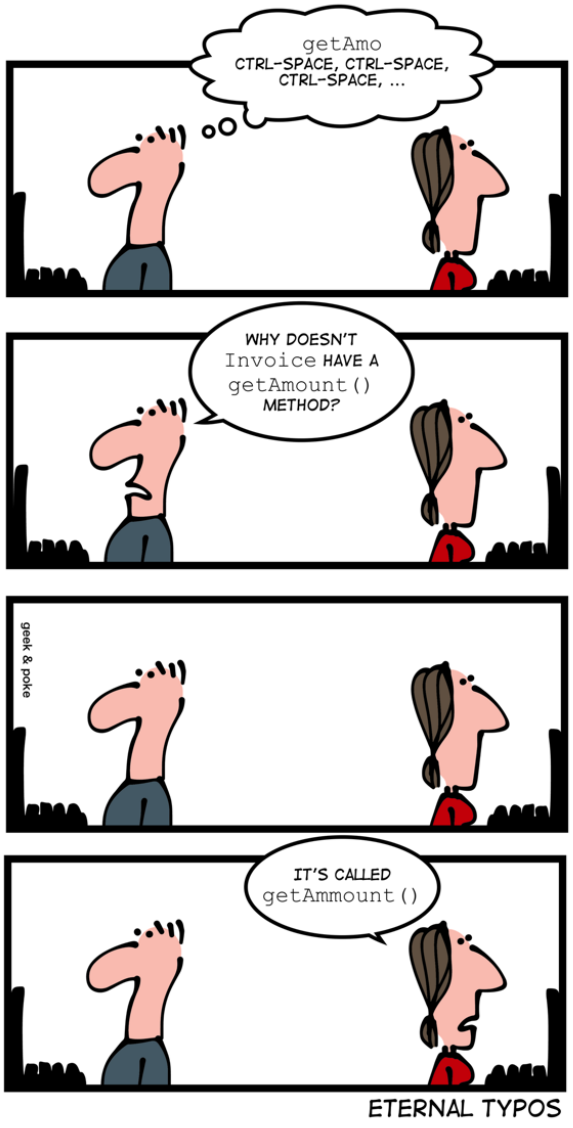
Internal (programmer's): *how?*

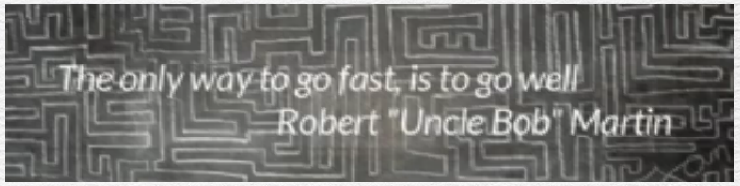
indico.cern.ch/event/34666/session/11/contribution/13/material/.../0.ppt

User's frustration

« Après avoir cliqué sur toutes les options et navigué dans tous les menus, je ne parviens toujours pas à savoir comment atteindre <ceci>"

"Pourquoi est-ce que je reçois ce message d'erreur? Je n'ai rien fait de mal - logiciel stupide!"

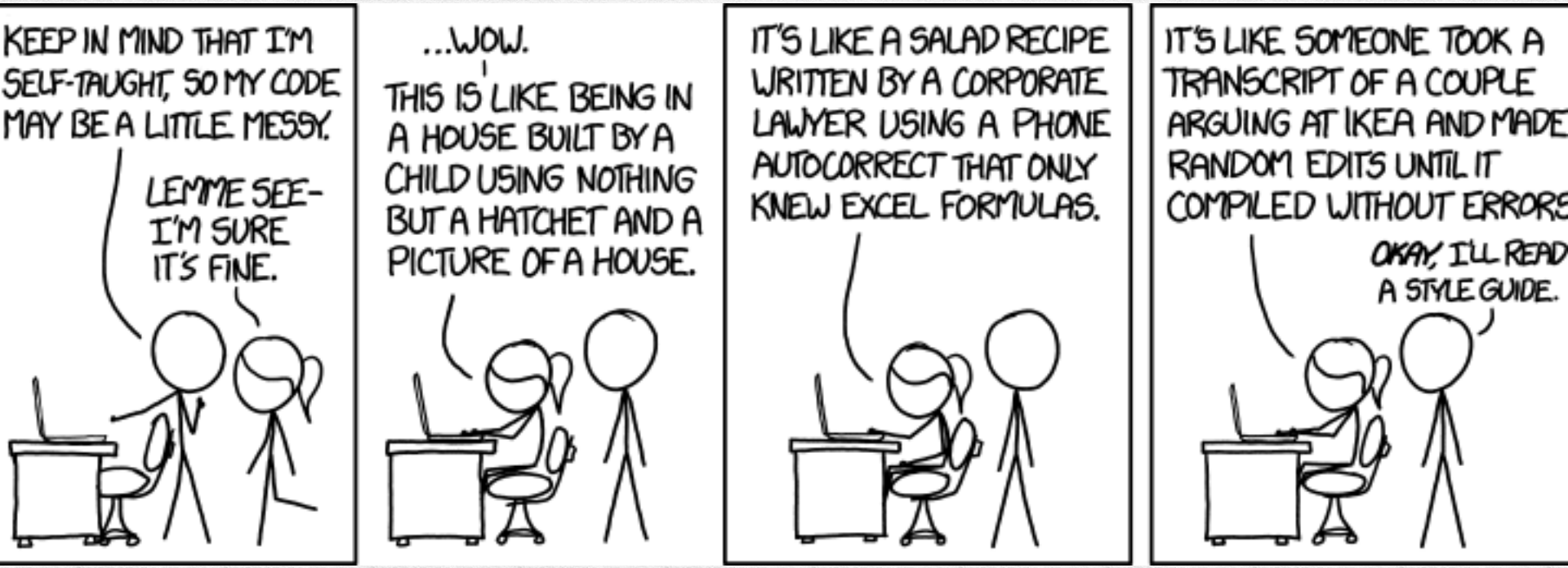




Programmer's frustration

"Je dois ajouter cette fonction, mais il est impossible de trouver un seul endroit où l'addition devrait figurer dans la structure de code complexe »

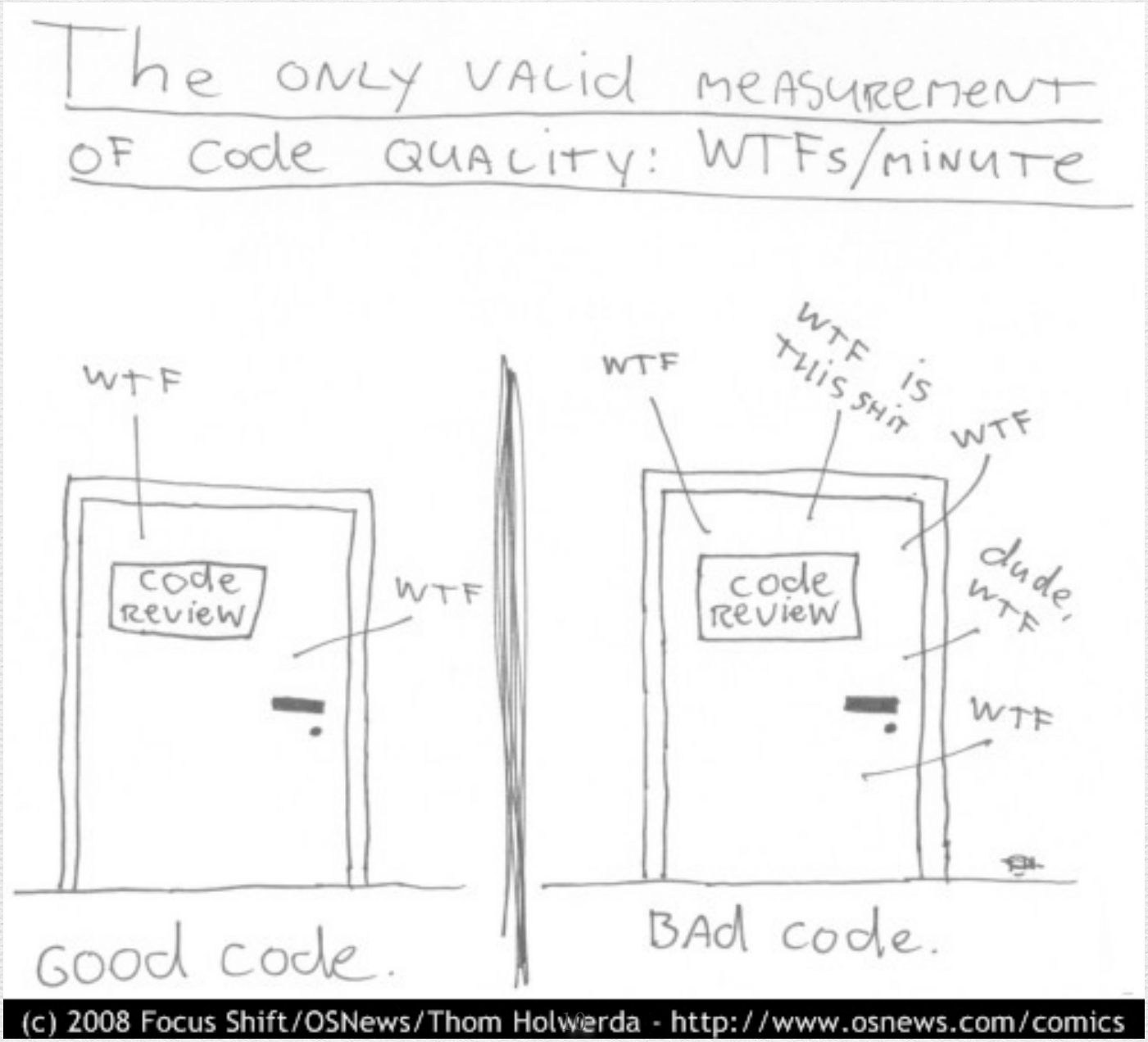
"Le programme produit une sortie incorrecte, mais après plusieurs heures passées à parcourir le code, je ne peux pas identifier les affirmations coupables"

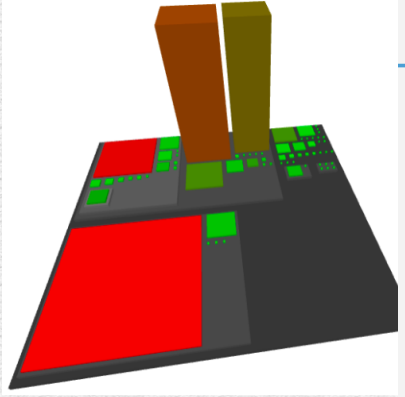

KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE- I'M SURE IT'S FINE.
...WOW.
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.
IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS
OKAY, I'LL READ A STYLE GUIDE.

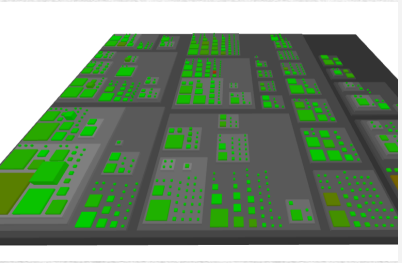
Source: XKCD

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





FitnesseBob

January 13, 2017 4:20 PM Version not provided

Issues Measures Code Administration

Quality Gate Passed

Bugs & Vulnerabilities

644 E Bugs

175 D Vulnerabilities

Leak Period: last 30 days
started 2 hours ago

0 New Bugs

0 New Vulnerabilities

Code Smells

22d A Debt

started 2 hours ago

1.5k Code Smells

4h New Debt

17 New Code Smells

Coverage

0.0% Coverage

2.8k Unit Tests

— Coverage on New Code

49k Lines of Code

Java 40k
JavaScript 8.1k
Web 413

Quality Gate
(Default) SonarQube way

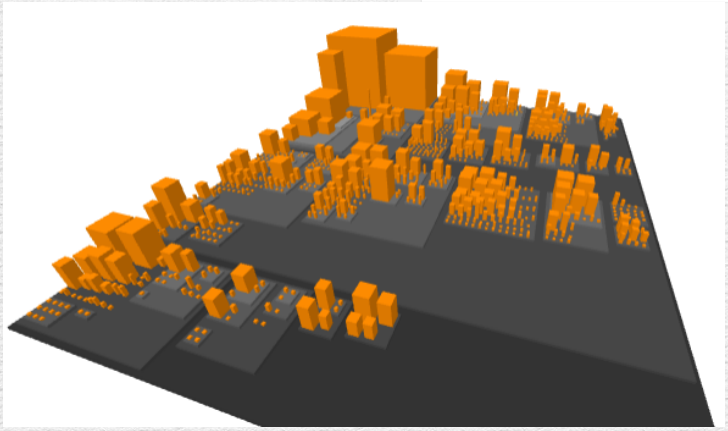
Quality Profiles
(Java) Sonar way
(JavaScript) Sonar way
(Web) Sonar way

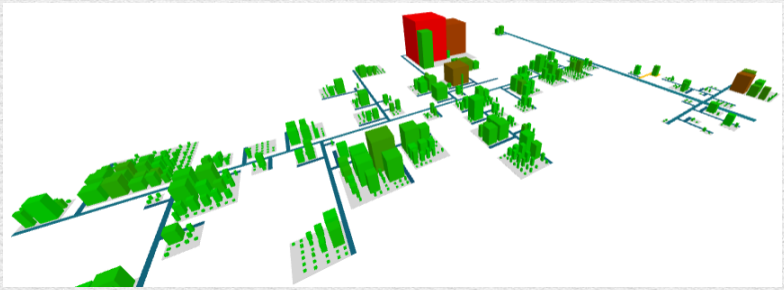
Key
FitnesseBob

Events All

Version: not provided
January 13, 2017

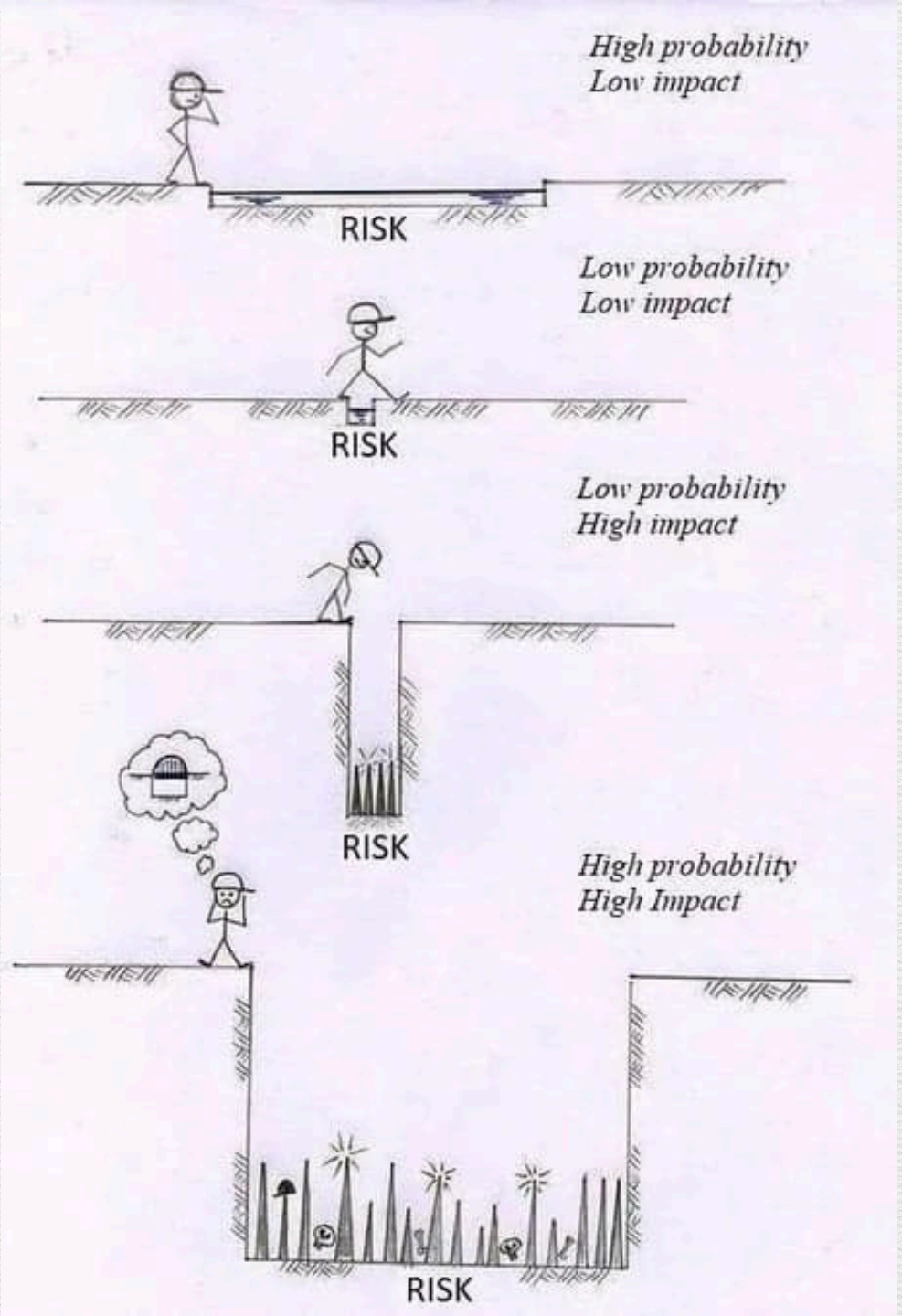
Quality Profile: Stop using 'Sonar way' (XML)
January 13, 2017





Une question de métrique ?

RISQUES


High probability
Low impact
RISK
Low probability
Low impact
RISK
Low probability
High impact
RISK
High probability
High Impact
RISK

La dette technique

Le fait de reporter l'écriture propre d'un code engendre **un coût qui augmentera avec le temps : c'est la dette technique.**

http://promyze.com/wp-content/uploads/2016/06/LaDetteTechnique.pdf

La dette technique

Côté client

Côté développeur

lesjoiesducode.fr

http://lesjoiesducode.fr/p

Exemples de dettes techniques

- **Une condition 'if' codée mais sans que le cas 'else' n'ait été traité.**
  - parfois un message de type 'TODO' dans le code.
  - S'il attend trop, c'est plus difficile
  - C'est encore plus difficile pour un autre développeur.
  - Cela peut devenir la source d'un bogue

http://promyze.com/wp-content/uploads/2016/06/LaDetteTechnique.pdf

Exemples de dettes techniques

- **La dette d'architecture** dans l'exploitation de bibliothèques ou de frameworks techniques obsolètes qui ont une faible plus-value. Changer ces framework constitue alors un cas classique de dette technique d'architecture.

http://promyze.com/wp-content/uploads/2016/06/LaDetteTechnique.pdf

Coût de la dette

- ++ **Correction :**
  - ✓ reporter des actions de développement dont le coût ne fait qu'augmenter. Plus on attend, plus il sera difficile de corriger la dette et plus le coût principal sera important.
- **Coût risqué** de la dette :
  - ✓ causé par l'impact négatif d'une dette.
  - ✓ difficile à évaluer…
- **Coût réel** :
  - ✓ Somme entre le coût principal (Correction/Réparation) et le cout risqué (coût des dégâts)
- Tout industriel souhaite en effet ***minimiser le coût réel*** de la dette. Le challenge est donc de payer le moins possible du coût principal en se préservant du coût risqué.

http://promyze.com/wp-content/uploads/2016/06/LaDetteTechnique.pdf

Stratégies pour faire face à la dette

1. Stratégie de la **dette non assumée** (pas même d'estimation du coût principal)
   - Paiement quand on voit les effets négatifs.. Suicidaire
2. Stratégie de la **dette par niveau de risque**
   - Paiement régulier en fonction des risques et du seuil à atteindre : suppose de savoir évaluer le risque !
3. Stratégie de la **dette par analyse du coût risqué**
   - Paiement régulier en se focalisant sur les parties à fort coût risqué (Conseillée)
4. Stratégie de la **non dette**
   - Paiement de l'intégralité du coût principal.
   - La moins risquée mais n'est peut-être pas optimale

Qu'est-ce que la qualité ?

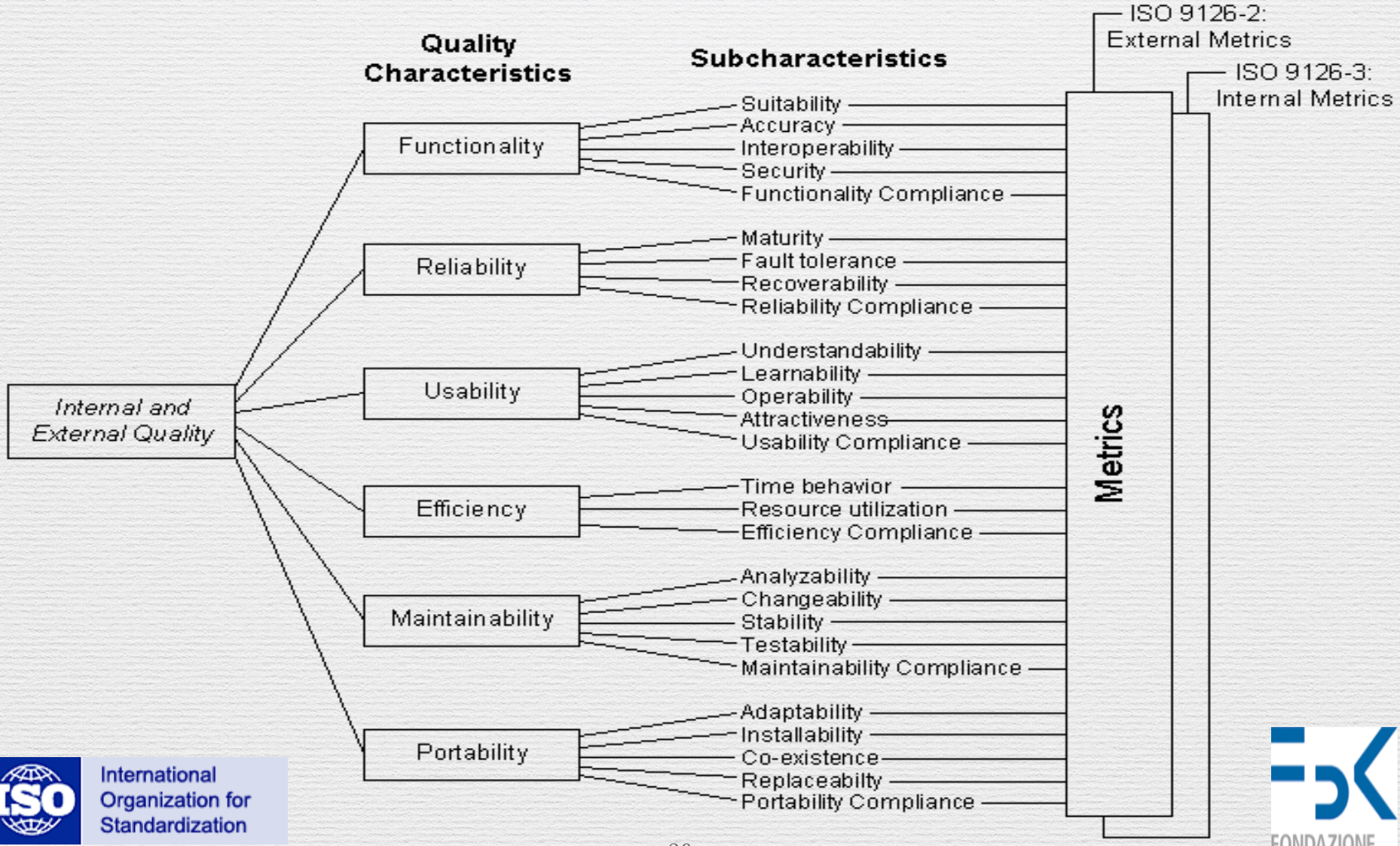
appréciation globale d'un logiciel,

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)



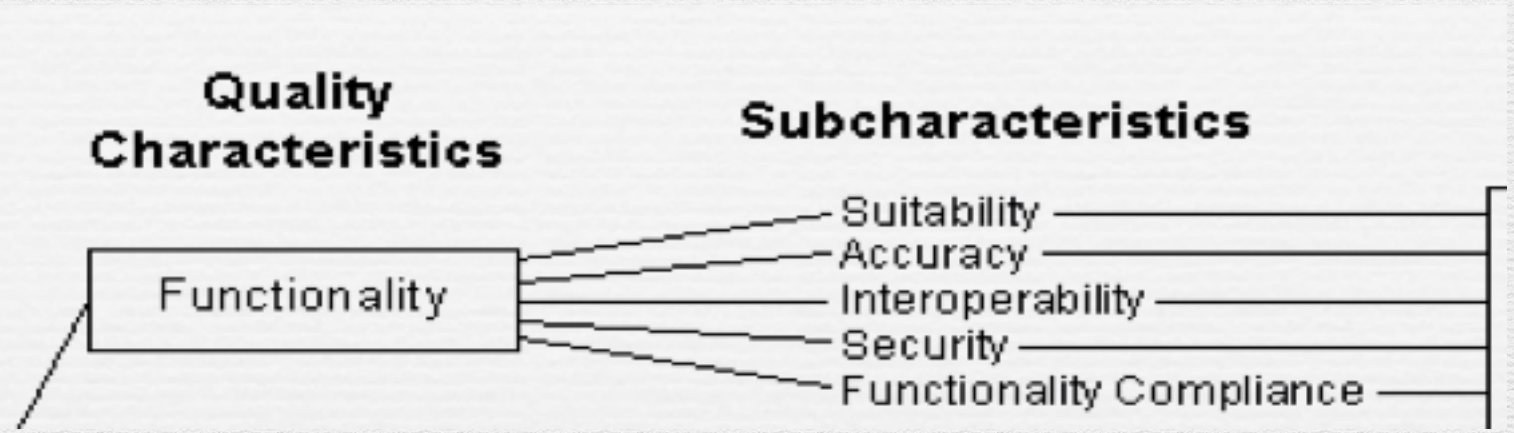
ISO International Organization for Standardization

FONDAZIONE BRUNO KESSLER

ISO/CEI 9126

➡ **Capacité fonctionnelle** (*functionality*)

✓ Répond-il aux besoins de ses utilisateurs ?



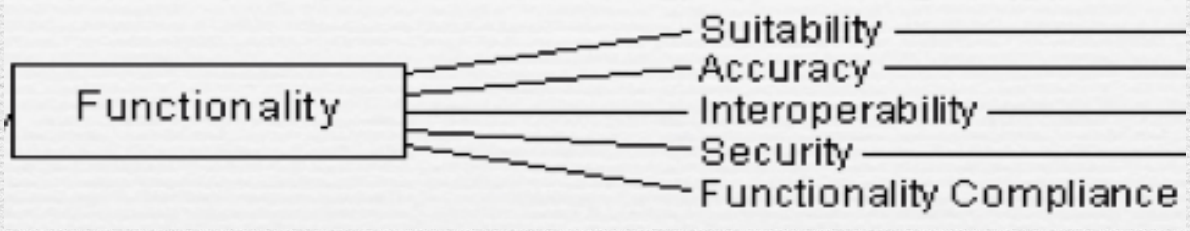
Capacité fonctionnelle

- 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

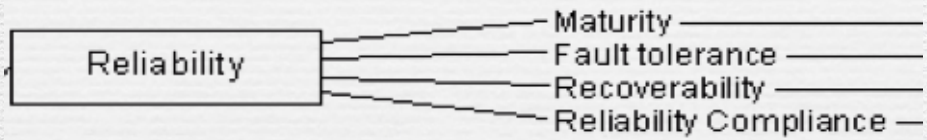
Aptitude : présence et adéquation d'une série de fonctions pour les tâches données
Exactitude : résultats ou effets justes ou convenus
Interopérabilité : interactions avec d'autres systèmes
Sécurité : accès non autorisé (accidentel ou délibéré) aux programmes et données



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

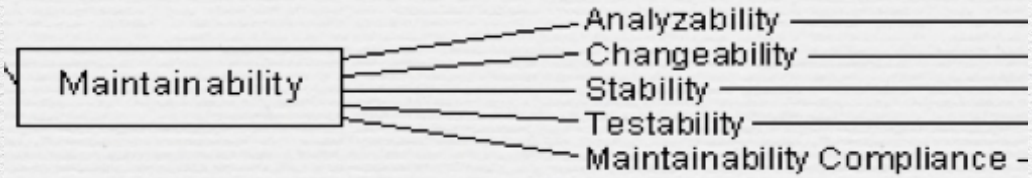
- Sous-caractéristiques

– **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

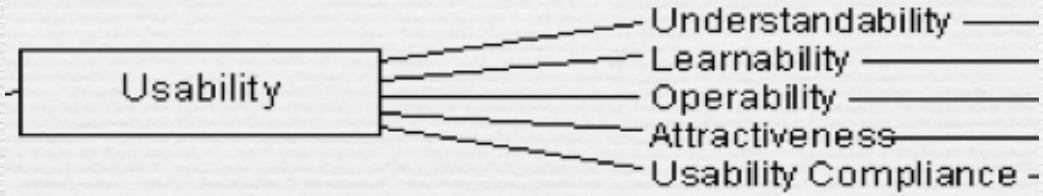
–**Facilité d'analyse** : effort nécessaire pour diagnostiquer les déficiences et leurs causes ou pour identifier les parties à modifier
–**Facilité de modification** : effort nécessaire pour modifier, remédier aux défauts ou adapter à l'environnement
–**Stabilité** : risque des effets inattendus des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

ISO/CEI 9126

➡ **Facilité d'utilisation** (*usability*)

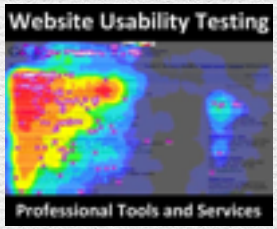
✓ Peut-il être utilisé à moindre effort ?

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



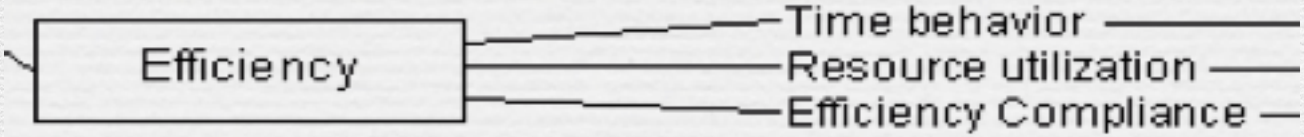
➡ Sous-caractéristiques

–**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

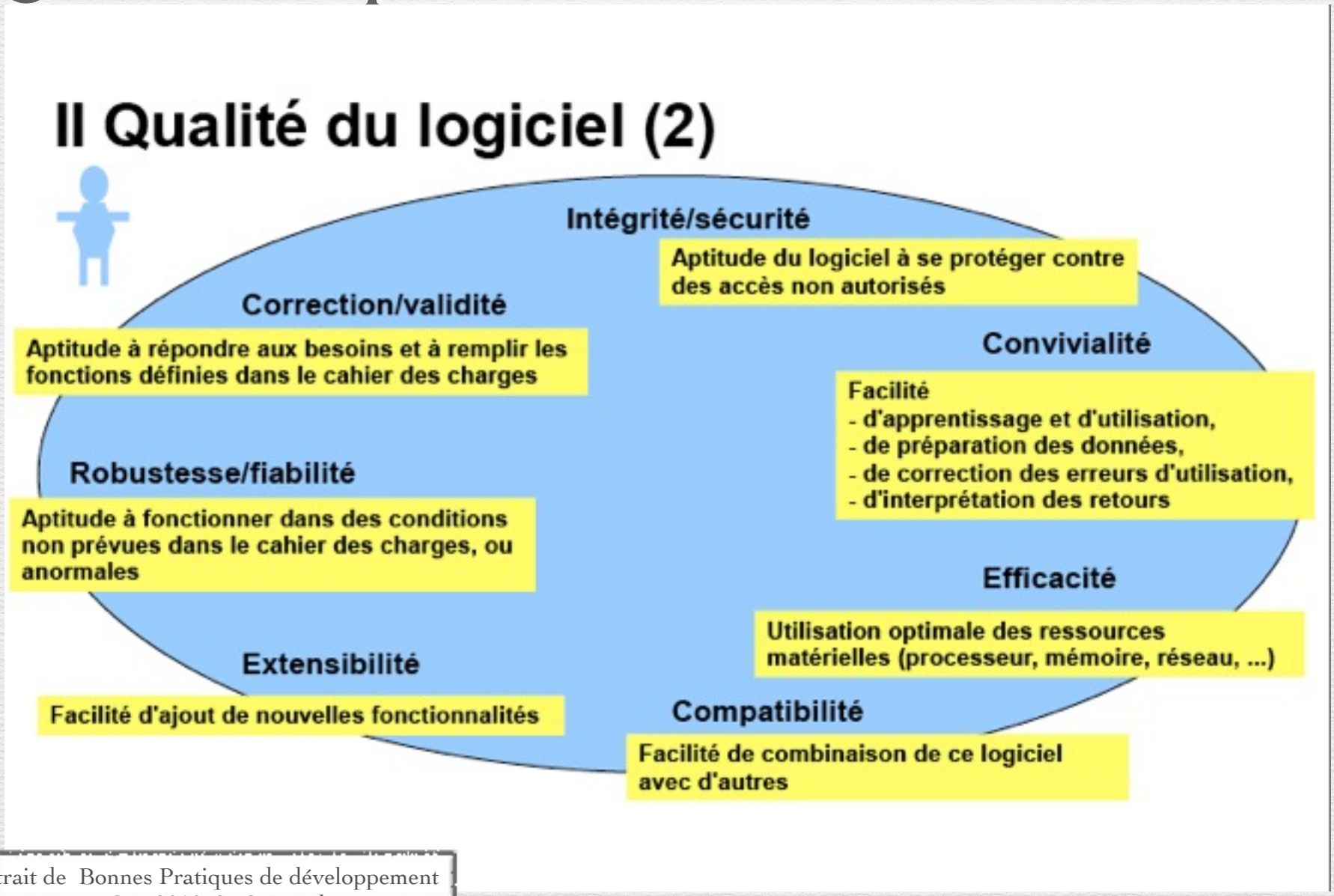
- Définitio…

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

➡ Sous-caractéristiques

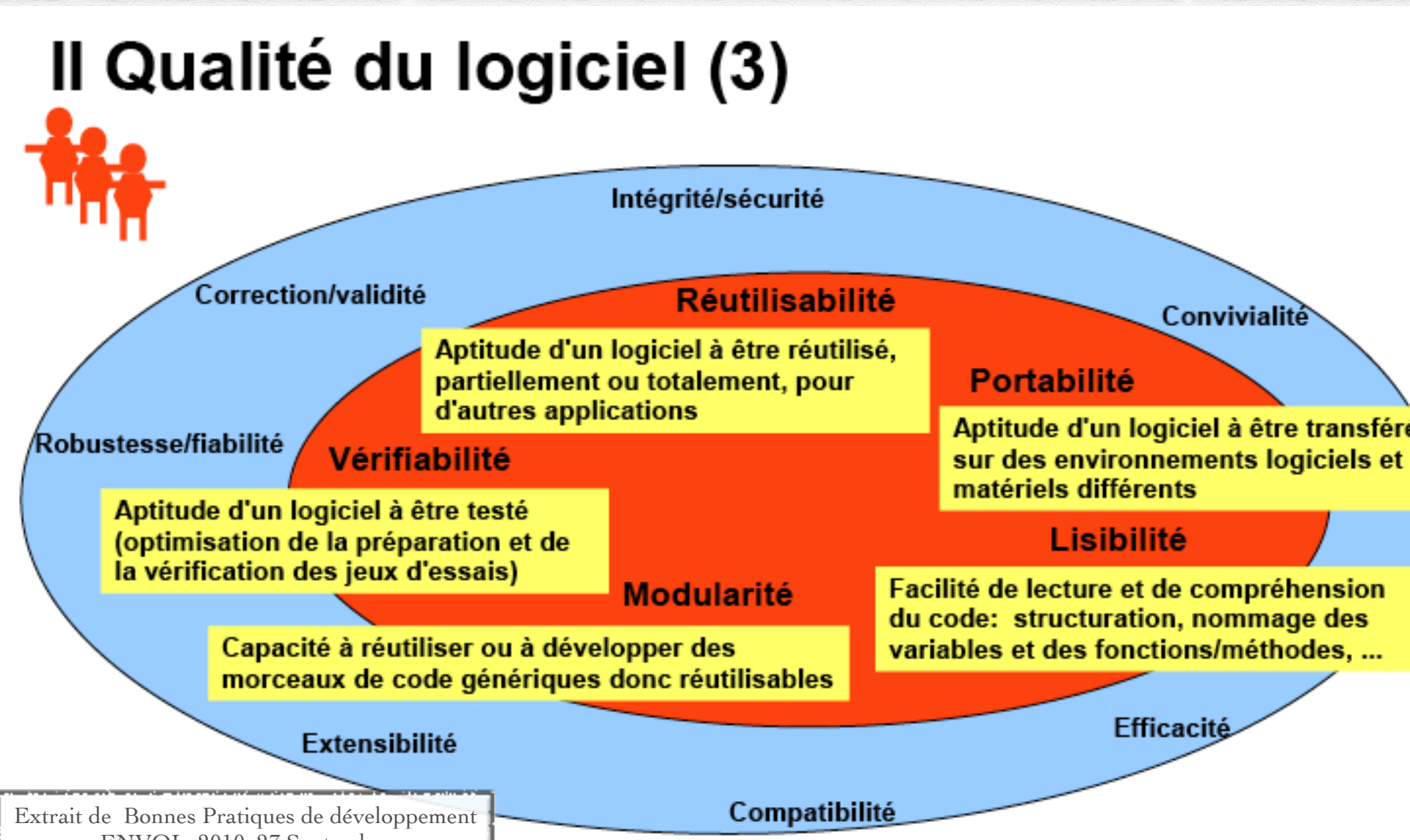
– **Facilité d'adaptation** : possibilité d'adaptation à différents environnements donnés sans que l'on ait recours à d'autres actions ou moyens que ceux prévus à cet effet par le logiciel.
– **Facilité d'installation** : effort nécessaire pour installer le logiciel dans un environnement donné.
– **Conformité 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



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

Qualité : point de vue Concepteur



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

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.

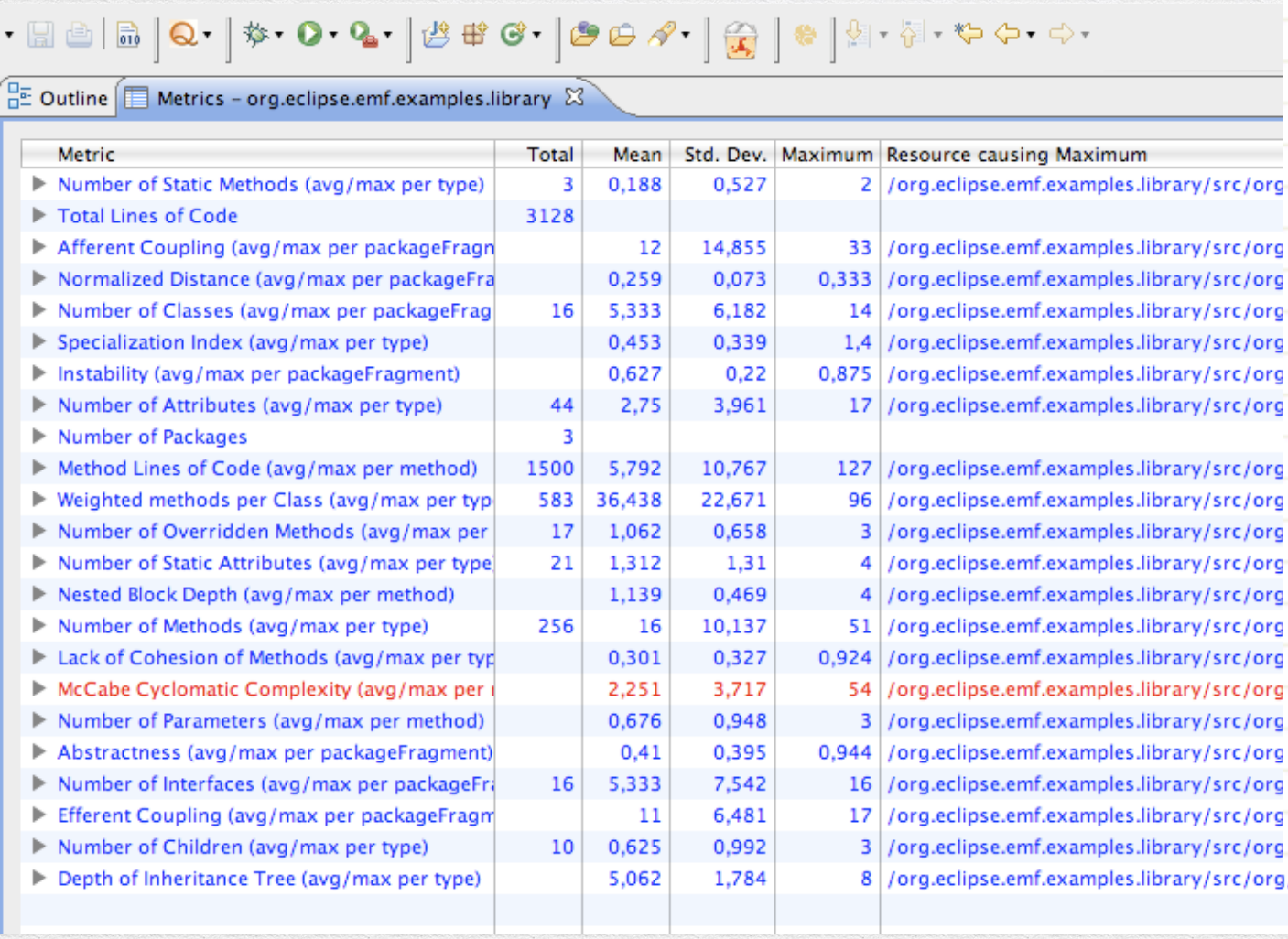
Exemples de métriques

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

➡ **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



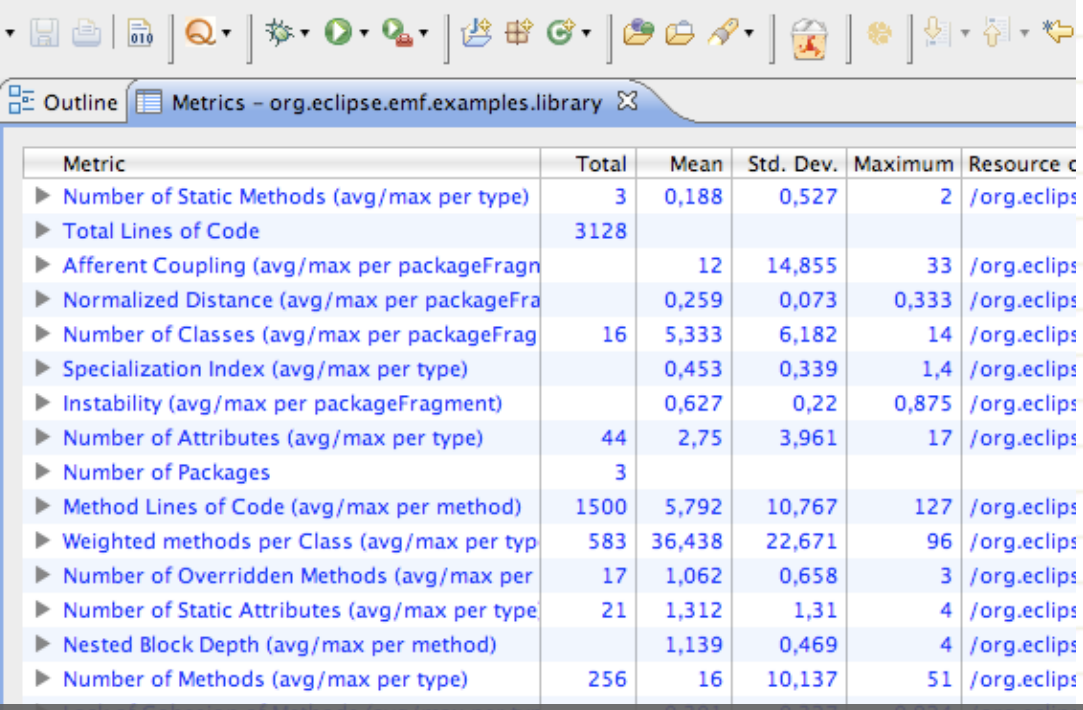
Outline | Metrics - org.eclipse.emf.examples.library

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 packageFragm		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



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

```
public class Identifiable implements Comparable<Identifiable> {
  private final String ident;
  /**
   * @return le hashCode de l'identifiant de cet objet.
   */
  @Override
public final int hashCode() {
    if (ident == null)    return 0;
    else return ident.hashCode();
  }
  /**
   * @return le rÈsultat de equals sur les identifiants.
   */
  @Override
public final boolean equals(Object o) {
    if (!(o instanceof Identifiable)) {
      return false; }
    String oid = ((Identifiable) o).ident;
    if (ident == null)
      return oid == null;
    else
      return ident.equals(oid);
  }
 @Override
public String toString() {
    if (ident == null)
      return "[null]";
    else
      return "[\"" + ident + "\"]";
  }


.................
```

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

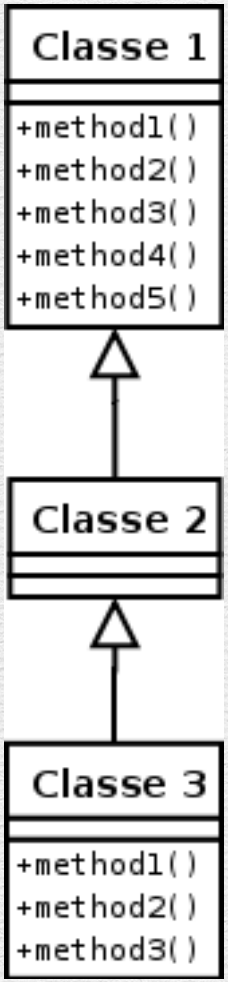
- public class Identifiable implements Comparable<Identifiable>

  -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

Indice de spécialisation

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

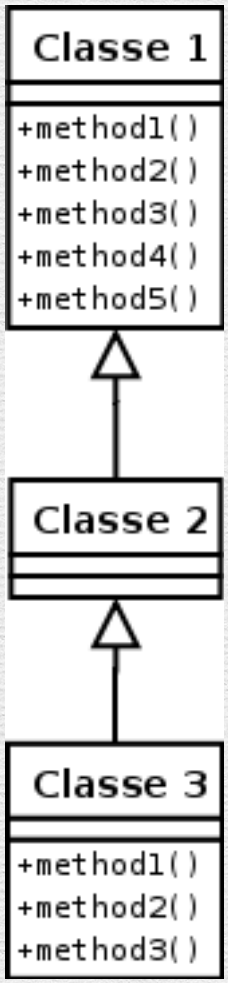
Augmente quand

- Le nombre de méthodes surchargées 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



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

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.

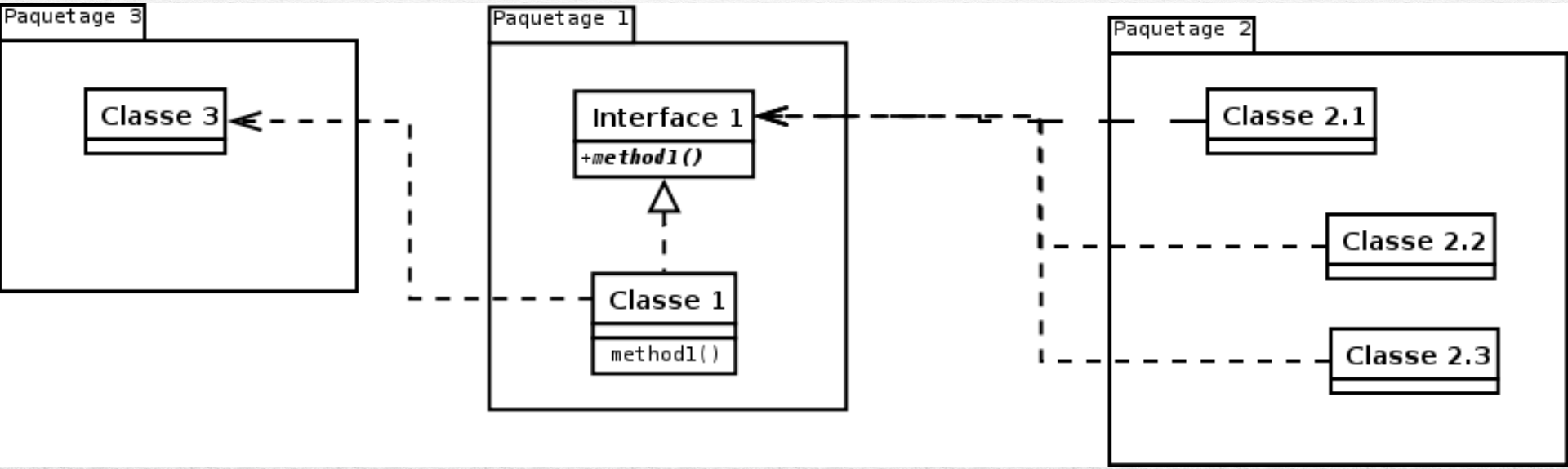
Instabilité = 1

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

GrapheX => 0;

RéseauSocial :
1 utilisation de l'extérieur dans test;
les 3 classes du package dépendent de classes extérieures:
Instabilité : 0,75

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

Indice d'instabilité - interprétation

- Indice compris entre 0 et 1 :

– 0 : le paquetage est stable.

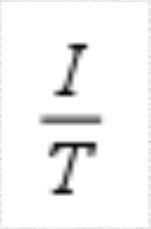
– 1 : le paquetage n'est pas stable.

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

- 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

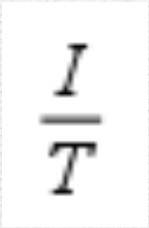
- Se calcule sur un paquetage ou un ensemble de paquetages.



- 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

$$|Abstractness + Instability - 1|$$

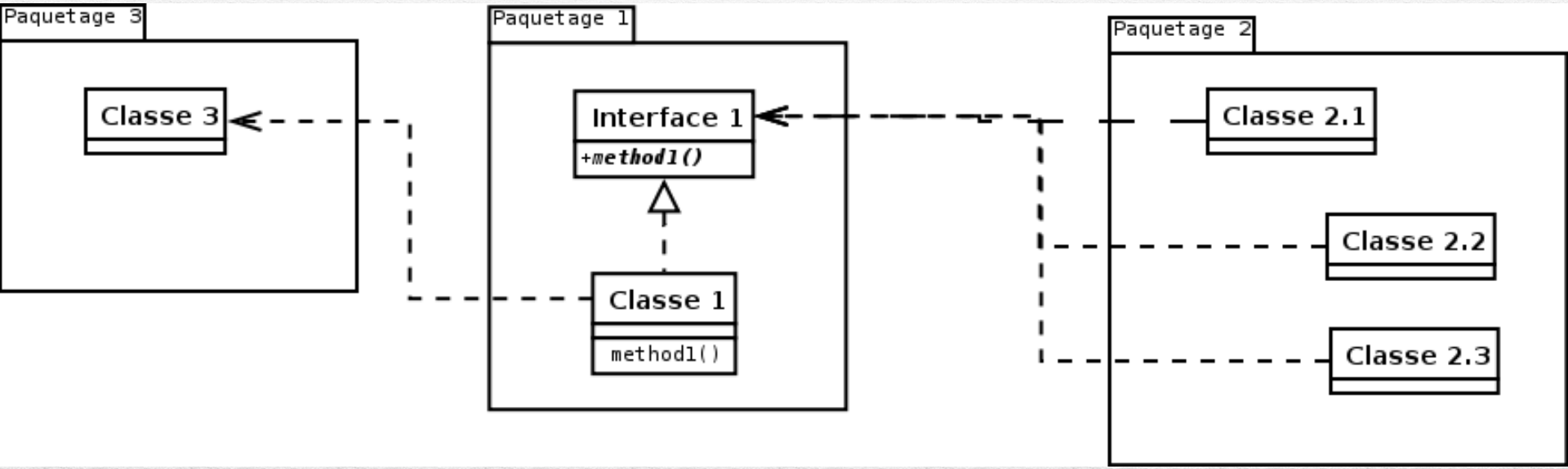
- Avec
- – Abstractness : l'indice d'abstraction du paquetage.
- – Instability : l'indice d'instabilité du paquetage.

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

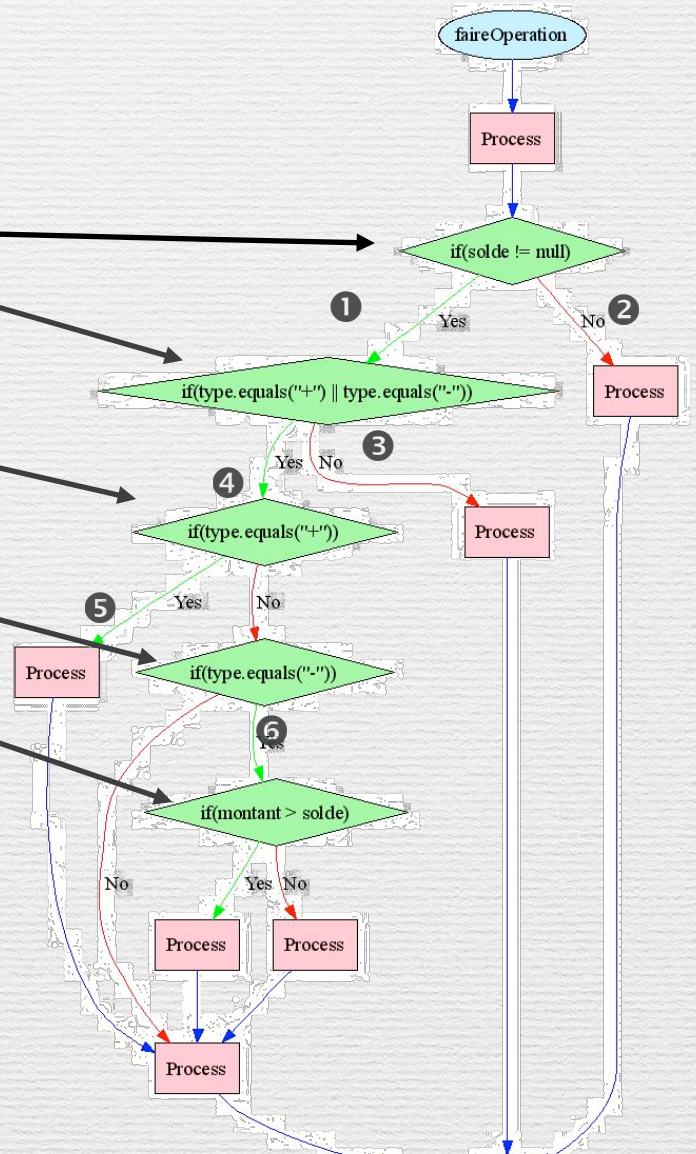
  - Une méthode avec une haute complexité cyclomatique est plus difficile à comprendre et à maintenir
  - > à 30 : refactoriser la méthode
  - < à 30 : acceptable si suffisamment de tests

  - La complexité cyclomatique est liée à la notion de « couverture de code"
    - Nombre de tests unitaires = sa complexité cyclomatique
    - **Un chemin = un test**

```
private ArrayList<Chemin> chemins(Sommet origine, HashMap<Sommet,ArrayList<Arc>> dejaVu){

    ArrayList<Chemin> chemins = new ArrayList<Chemin>();

    if  (dejaVu.containsKey(origine)){
        chemins.add(new Chemin(dejaVu.get(origine)));
        return chemins;
    }


    dejaVu.put(origine, new ArrayList<Arc>());


    Collection<Arc<Sommet>> voisins = graphe.voisins(origine);
    HashMap<Sommet,ArrayList<Arc>> dejavVuLocal = new HashMap<Sommet, ArrayList<Arc>>();

    for (Arc<Sommet> a : voisins) {
        Sommet destination = a.destination();
        dejavVuLocal= new HashMap<Sommet,ArrayList<Arc>>(dejaVu);

        if (nouvelleDestinationOuNouvelArcSansRetour(origine,dejavVuLocal,destination,a)) {
            dejavVuLocal.get(origine).add(a);
            ArrayList<Chemin> cheminsLocaux = chemins(destination,dejavVuLocal);
```

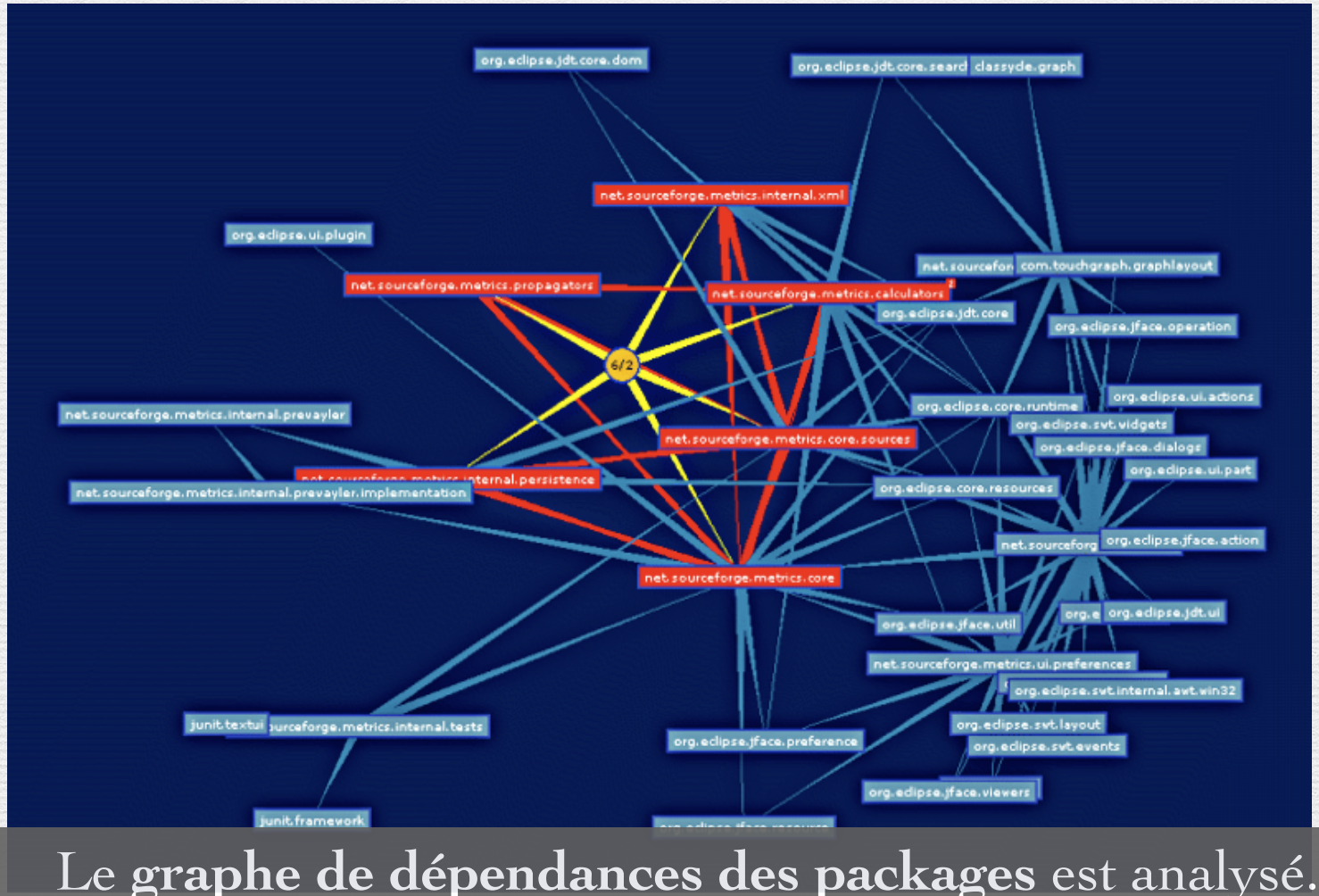
Complexité cyclomatique d'une méthode

```
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

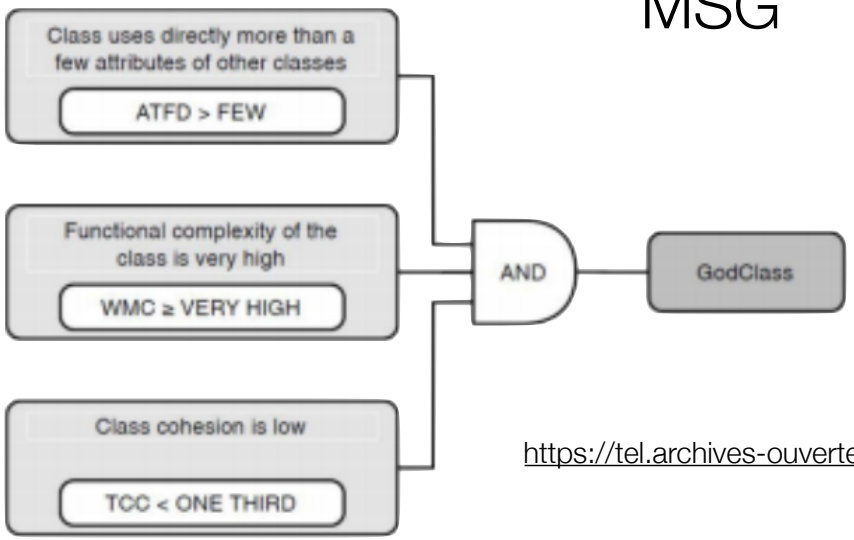
(plugin Metrics : http://metrics.sourceforge.net/)



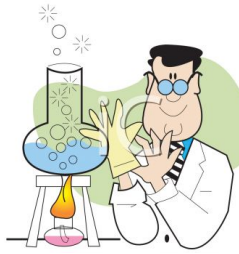
Le **graphe de dépendances des packages** est analysé.
Les packages fortement connectés sont colorés en rouge. Les autres packages ne participant pas à des cycles sont de couleur bleue.

Metrics compress the system into numbers

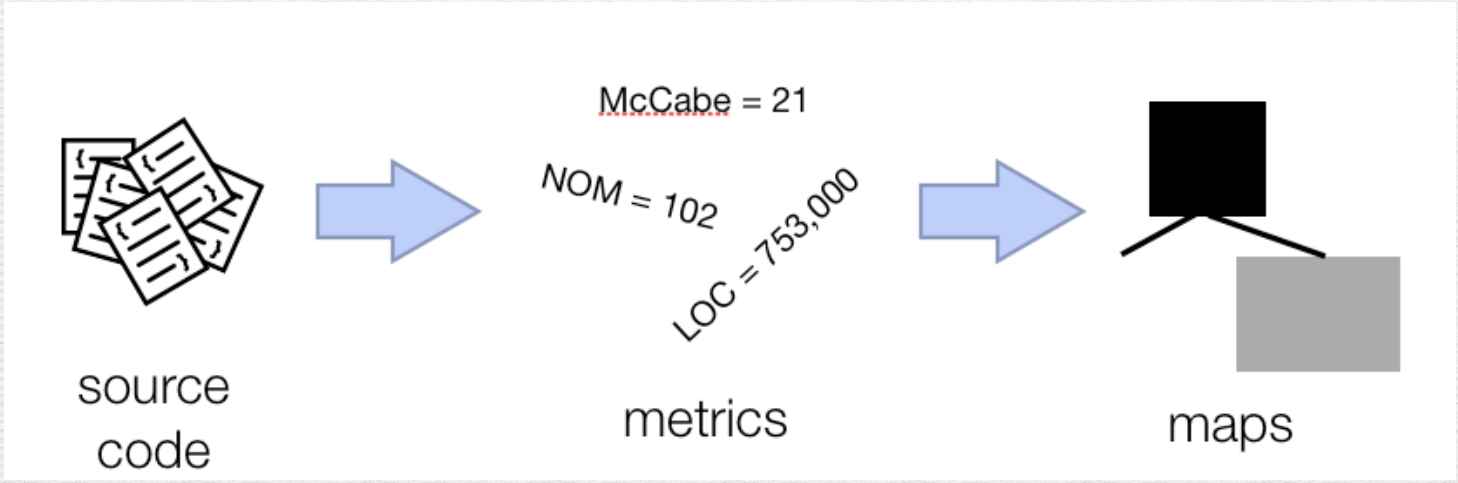
NOC	NOM	DUPLINES
NOCmts	LOC	NAI
NOPA	TCC	NOA
WLOC	WMC	NI
WNOC	CYCLO	...
WOC	ATFD	
MSG	HNL	

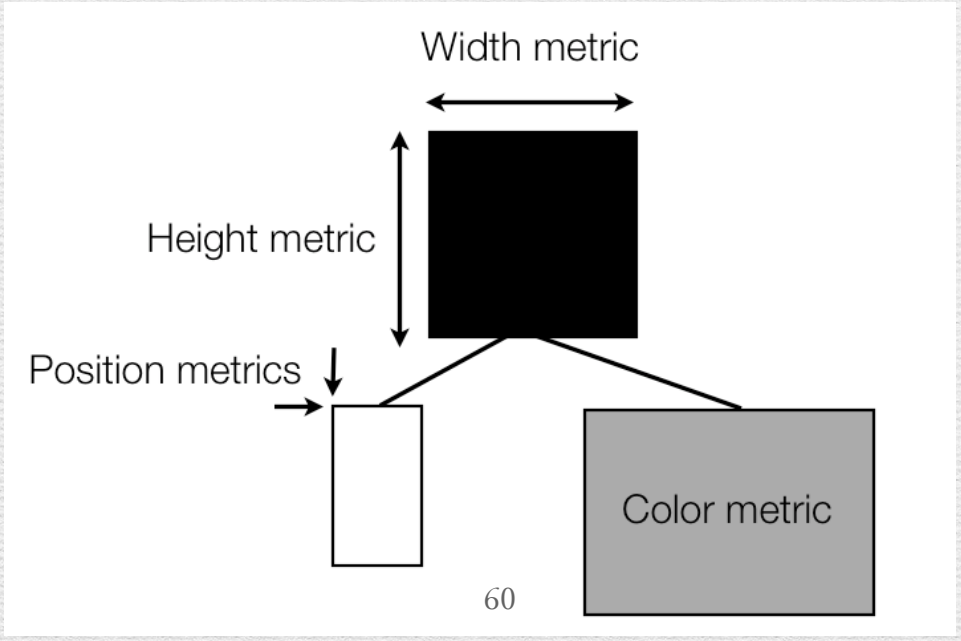


https://tel.archives-ouvertes.fr/tel-00790056/document

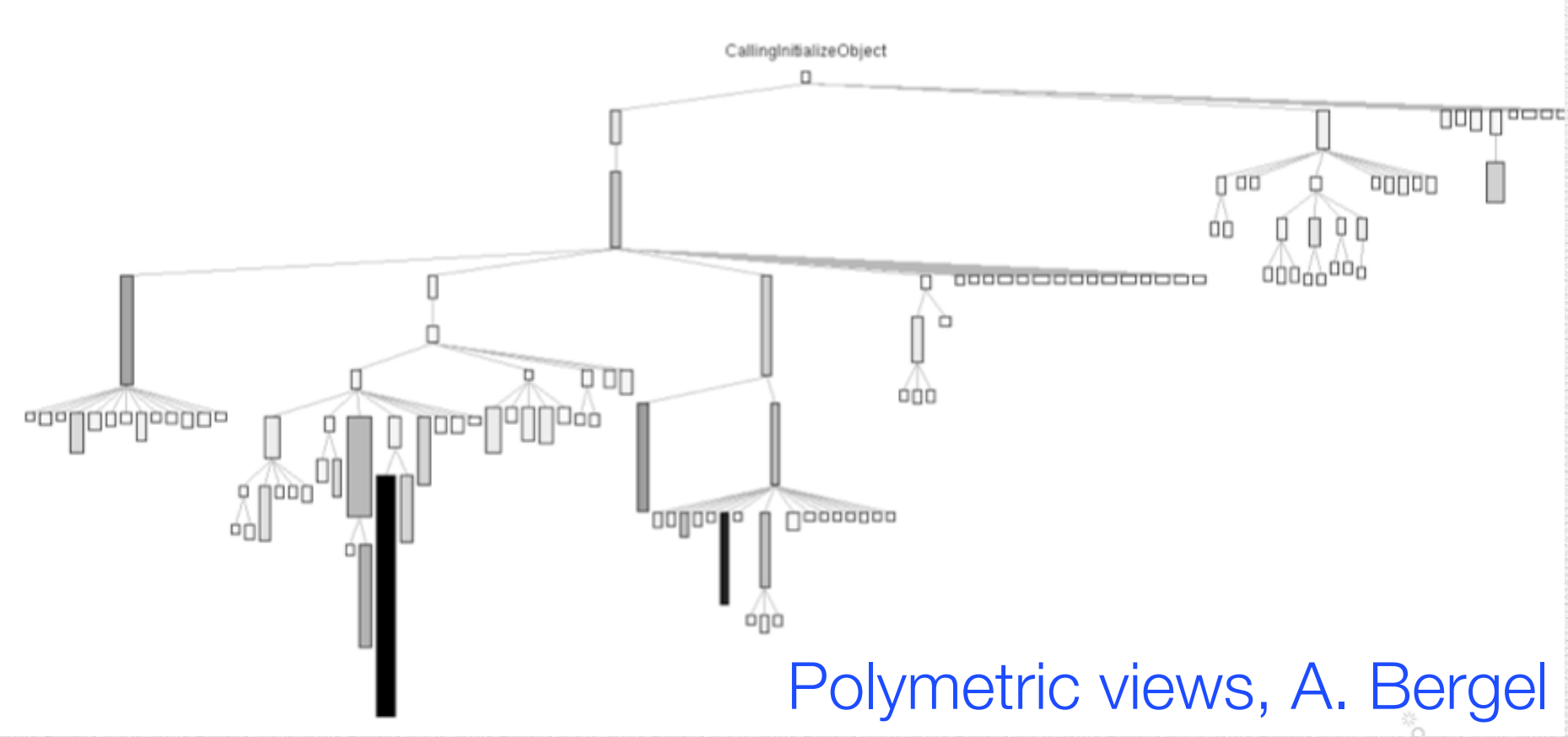


Tests statiques
Autres Visualisations





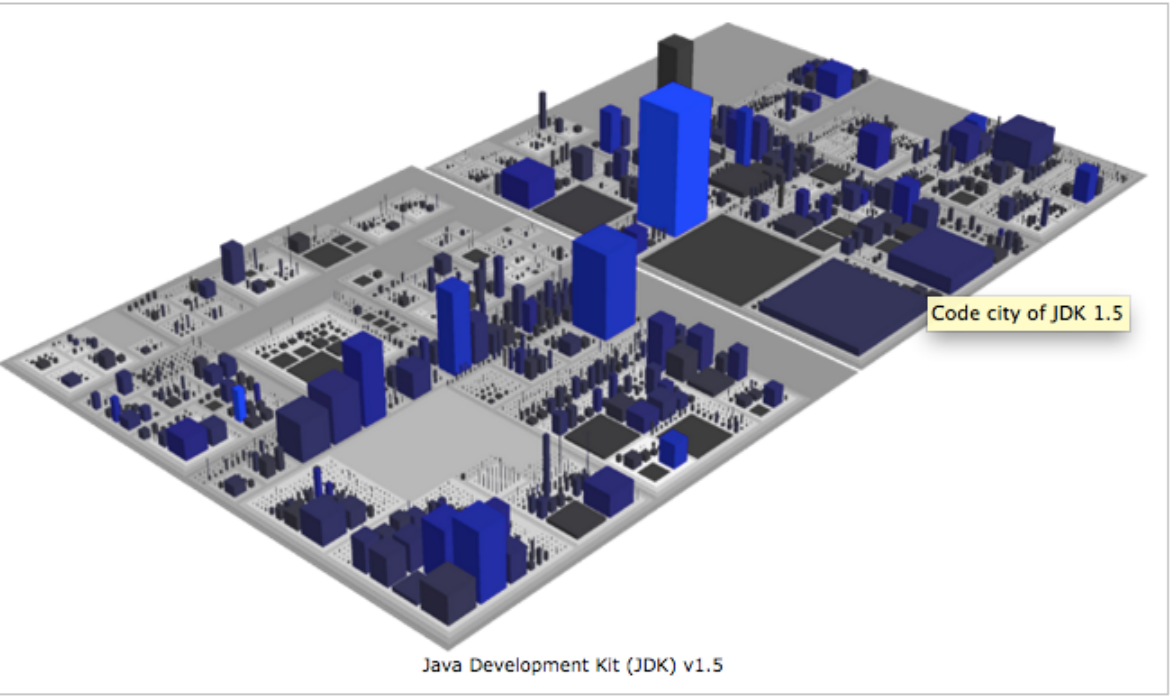
Tests statiques
Autres Visualisations

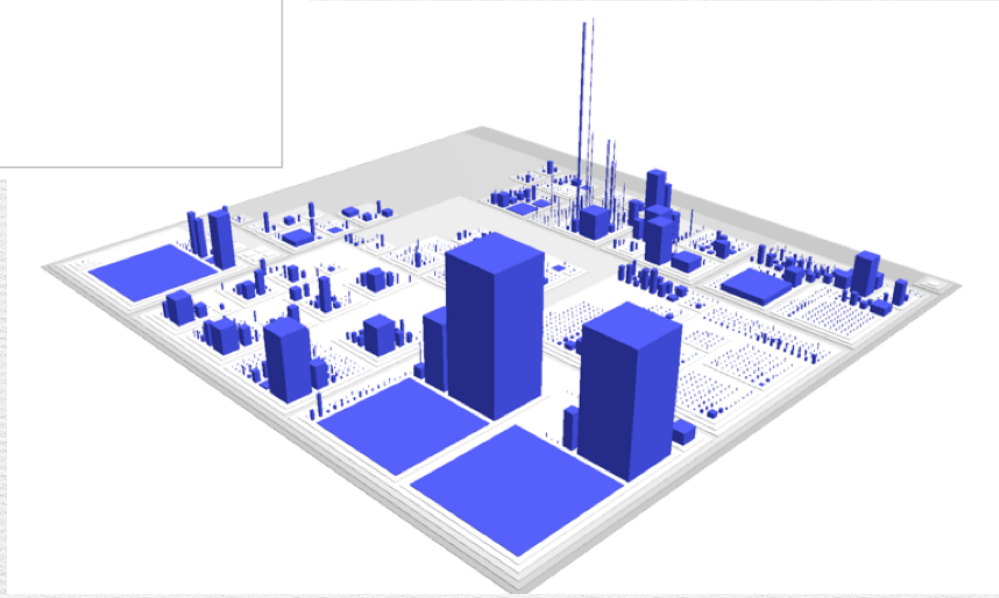


Polymetric views, A. Bergel

Tests statiques

Autres Visualisations





CodeCity (Wettel, Univ Lugano)

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

https://www.youtube.com/watch?feature=player_embedded&v=U2MJ3La2y4U

Couverture de code

- Un outil de couverture de code est un outil permettant de vérifier dynamiquement (à l'exécution) quelles parties du code source à tester ont effectivement ´et´e ex´ecut´ees (« couvertes » par le(s) test(s)).

Critères de couverture 1/2

Les principaux critères de couverture sont :

- Function Coverage : quelles fonctions/méthodes ont été exécutées ?
- Statement Coverage : quelles lignes du code source ont été exécutées ?
- Branch Coverage : quelles parties d'un bloc conditionnel (if/else) ont été exécutées ?
- Condition Coverage : quelles valeurs de decision (ex : i < num && ! skip) ont été rencontrées a l'execution ?
- Entry/Exit Coverage : toutes les possibilités d'invocation et de sortie d'une fonction/méthode ont-elles été utilisées ?
- Loop Coverage : aucune, une seule, plusieurs executions successives d'une boucle ?
- Path Coverage : tous les chemins d'execution d'une partie du code ont-ils été empruntés ?

http://www.normalesup.org/~labatut/ED6/cours-6.pdf

Critères de couverture (2/2)

Statement Coverage ⇏ *Condition Coverage* :

```
1  void m(int i) {
2      System.out.println("Hello");
3
4      if (i <= 0)
5          System.out.println(", World!");
6
7      System.out.println("!");
8  }
```

Si `m(-1)` est appelée dans un test unitaire, 100% de *Statement Coverage*, mais 50% de *Condition Coverage*.

dans Eclipse

MaisonNumerique	66,8 %	576	286	862
src	46,5 %	227	261	488
iut.maisonNumerique	0,0 %	0	187	187
iut.sensors	59,7 %	46	31	77
NonAccessibleSensorException.java	34,6 %	9	17	26
PhysicalSensor.java	72,5 %	37	14	51
memoryPK	70,1 %	54	23	77
iut.sensors.td	86,4 %	127	20	147
tests	93,3 %	349	25	374

Métriques


Johan den Haan a retweeté
Richard Dalton @richardadalton · 25 juil.
Too much software productivity is measured by how far the bullet is from the gun, rather than how close it is to the target.
À l'origine en anglais
3
142
199

Méthodologie de la production d'applications

Que savez-vous à présent ?

Mireille Blay-Fornarino
Décembre 2017
https://mbf-iut.i3s.unice.fr/

Service-Public.fr refondu en mode agile avec intégration continue

Le 16 Décembre 2016

CIO

- Entièrement refondu depuis un an, le site Service-Public.fr reçoit chaque mois en moyenne 20 millions de visites, avec des variations saisonnières. Sa refonte, entreprise début 2015 avec le concours d'Octo Technology, a été réalisée en 9 mois et l'évolution du site avec l'ajout de nouvelles fonctionnalités se poursuit maintenant en ***intégration continue***….

- Au programme : le pilotage d'un projet agile, les ***usagers au coeur de la conception*** du site, l'*animation en mode agile* d'une large équipe fonctionnelle, les pratiques pour assurer la *propriété collective* du code applicatif, les principes devops clés pour mettre « sereinement » **en production toutes les 2 semaines.** Le dernier point abordé, loin d'être le moindre mais souvent négligé dans les développements de sites web, concernait l'intégration native des normes d'accessibilité dans Service-public.fr.

http://www.cio-online.com/actualites/lire-service-publicfr-refondu-en-mode-agile-avec-integration-continue-8979.html?utm_source=email-cio&utm_medium=email&utm_campaign=Newsletter

1. Sélectionnez les attributs de qualité les plus importants.

Hiérarchisez en fonction du problème et mettez vous d'accord sur les moyens de les atteindre.

2. Responsabilisez et communiquez

Les supports de communication partagés; des outils de vérification automatique;

3. Réunissez -vous

- Daily & Rétrospective meetings

D'après https://codingsans.com/blog/ensure-code-quality-in-offshore-software-development