

Tests et excellence technique au cœur du développement agile



Isabelle BLASQUEZ
@iblasquez

Septembre 2016

Evolution des architectures logicielles

Ahaa : “The evolution of
#SoftwareArchitecture” bit.ly/1JPD1k1

 Voir la traduction

THE EVOLUTION OF SOFTWARE ARCHITECTURE

1990's

SPAGHETTI-ORIENTED
ARCHITECTURE
(aka Copy & Paste)



2000's

LASAGNA-ORIENTED
ARCHITECTURE
(aka Layered Monolith)



2010's

RAVIOLI-ORIENTED
ARCHITECTURE
(aka Microservices)



Dans l'esprit du
développement agile

WHAT'S NEXT?

PROBABLY PIZZA-ORIENTED ARCHITECTURE

Un développement agile s'articule autour :

→ d'un **Bon Produit (RIGHT PRODUCT)**

QUOI

→ **La spécification agile** permet d'identifier le bon produit, celui qui correspond aux attentes des utilisateurs.
Elle s'exprime sous forme de **User Story**

QUAND

→ **La priorisation et l'estimation** permettent de définir à quel moment du développement, quelle partie du produit (quoi) devra être livrée, et quelles tâches et combien de temps seront nécessaire pour effectuer cette livraison.

Trouver et écrire les US

→ d'un **Produit correctement implémenté (PRODUCT RIGHT)**

COMMENT

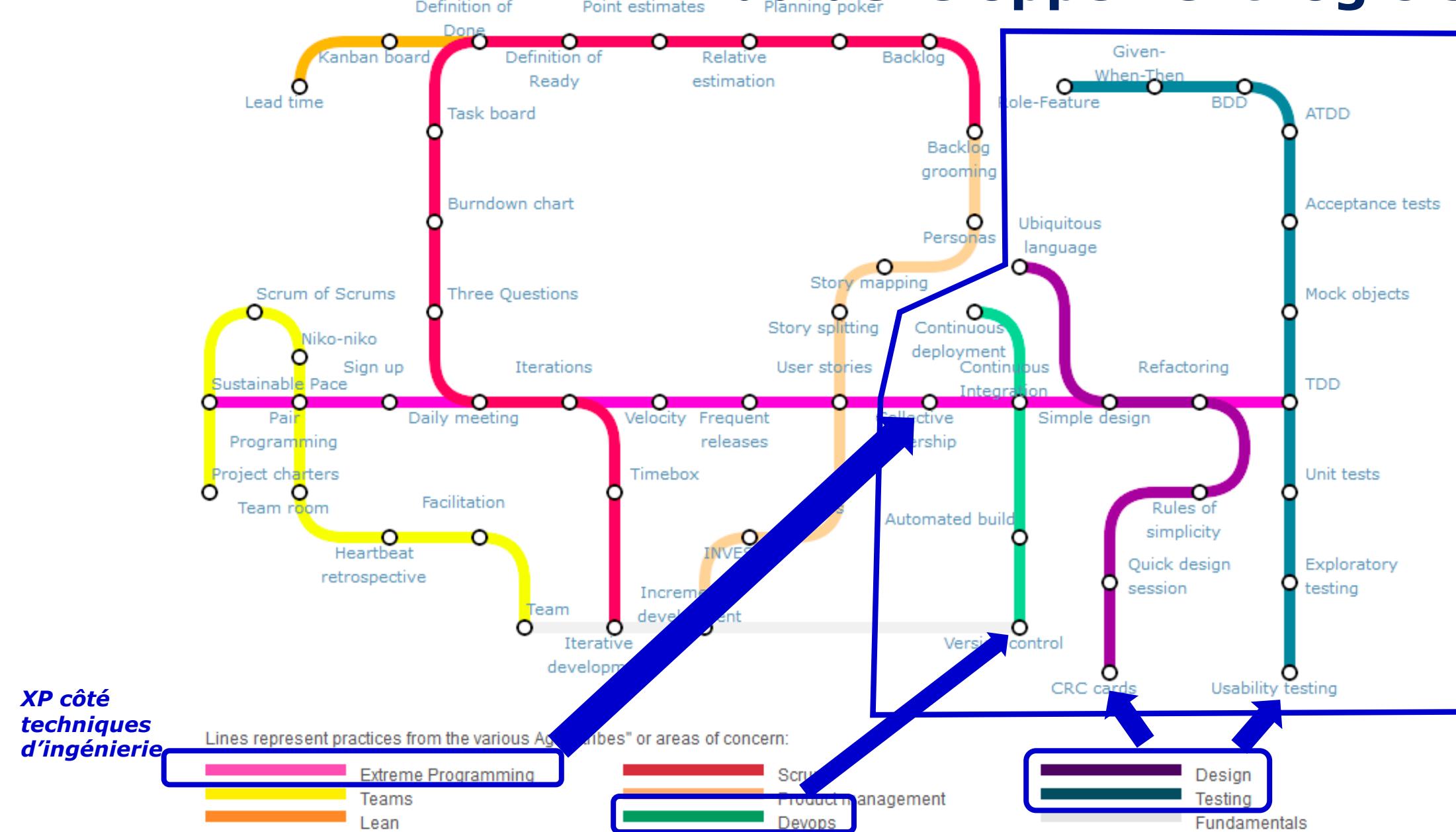
→ **Une mise en avant des tests** pour pouvoir développer le QUOI le plus proprement possible et en toute confiance

→ Une attention toute particulière portée à **l'excellence technique** en considérant le développeur comme un artisan du logiciel et en le (re)plaçant au cœur du développement logiciel.

Prioriser et estimer les stories

Implémenter les stories

Tests et excellence technique au cœur du développement logiciel agile



Extrait : <https://www.agilealliance.org/agile101/subway-map-to-agile-practices>



Le Test dans le développement logiciel

Le Test, c'est quoi ?

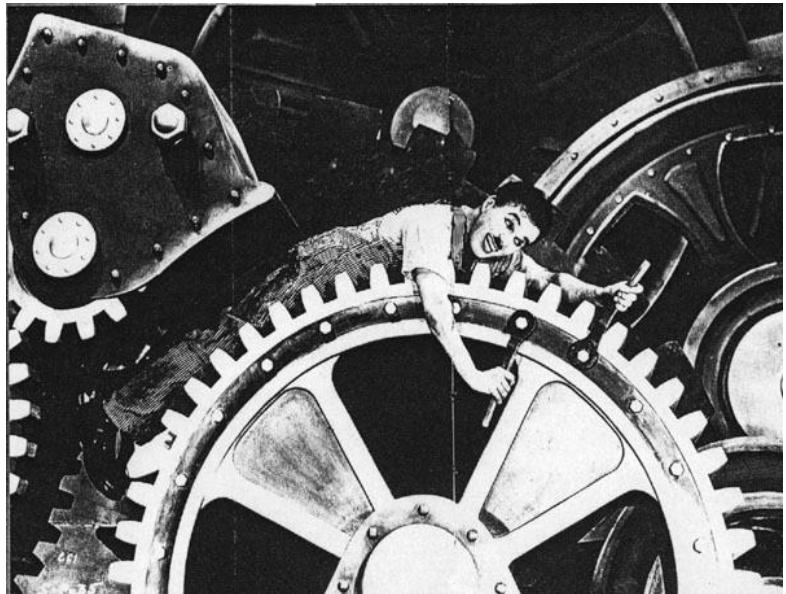
Le processus de test consiste à exécuter un programme dans l'intention de détecter des erreurs.

(Myers, Sandler, 2004)

***Tester peut seulement montrer la présence d'erreur,
mais pas leur absence.***

(Dijkstra, 1972)

Le test : pour quoi ? ... Un outil de qualité logicielle



Vérification

Fonctionnement correct du produit
(*Product Right*)



Validation

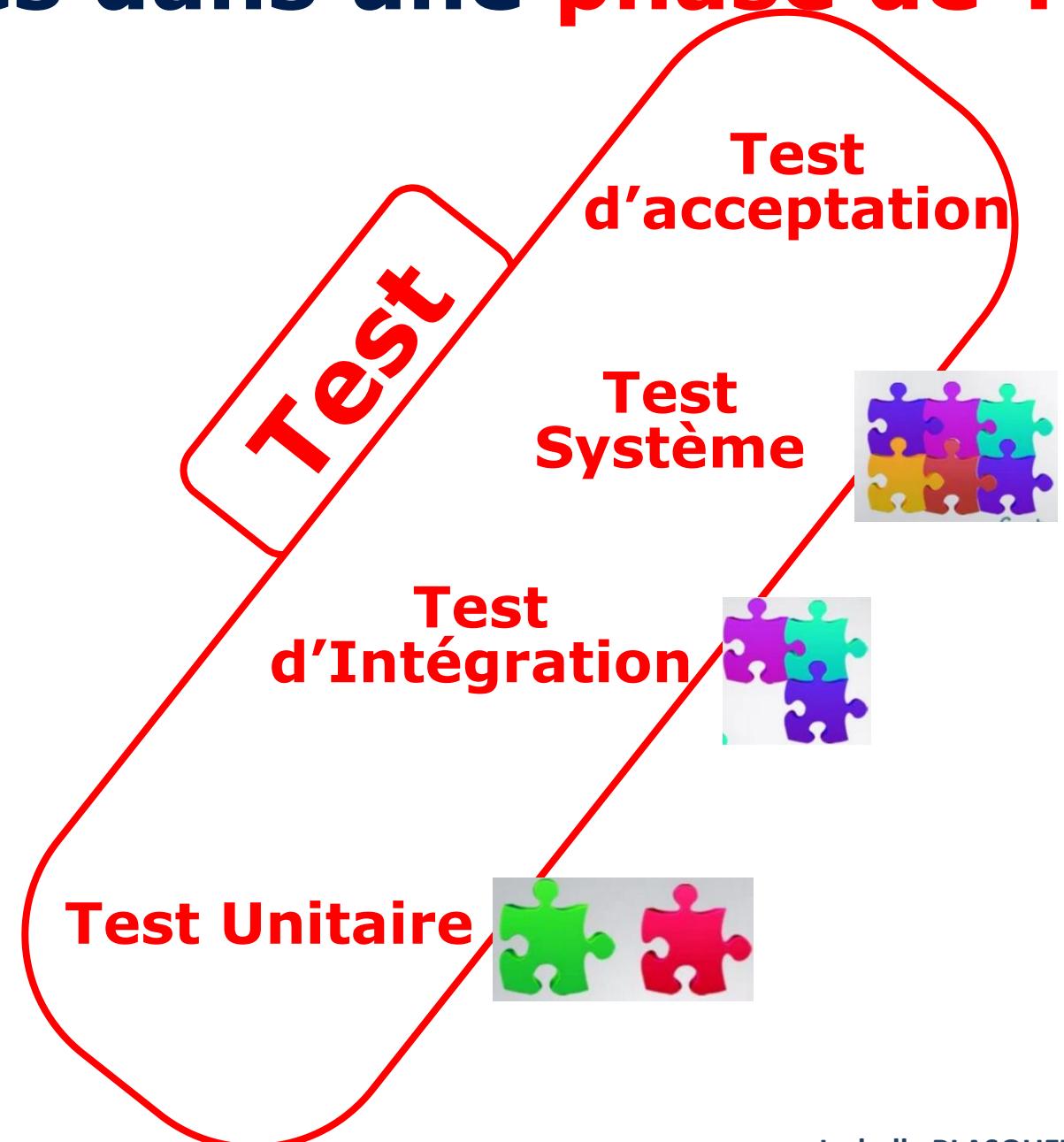
Respect des exigences utilisateurs
(*Right Product*)

Différentes granularités dans une phase de Test

Analyse

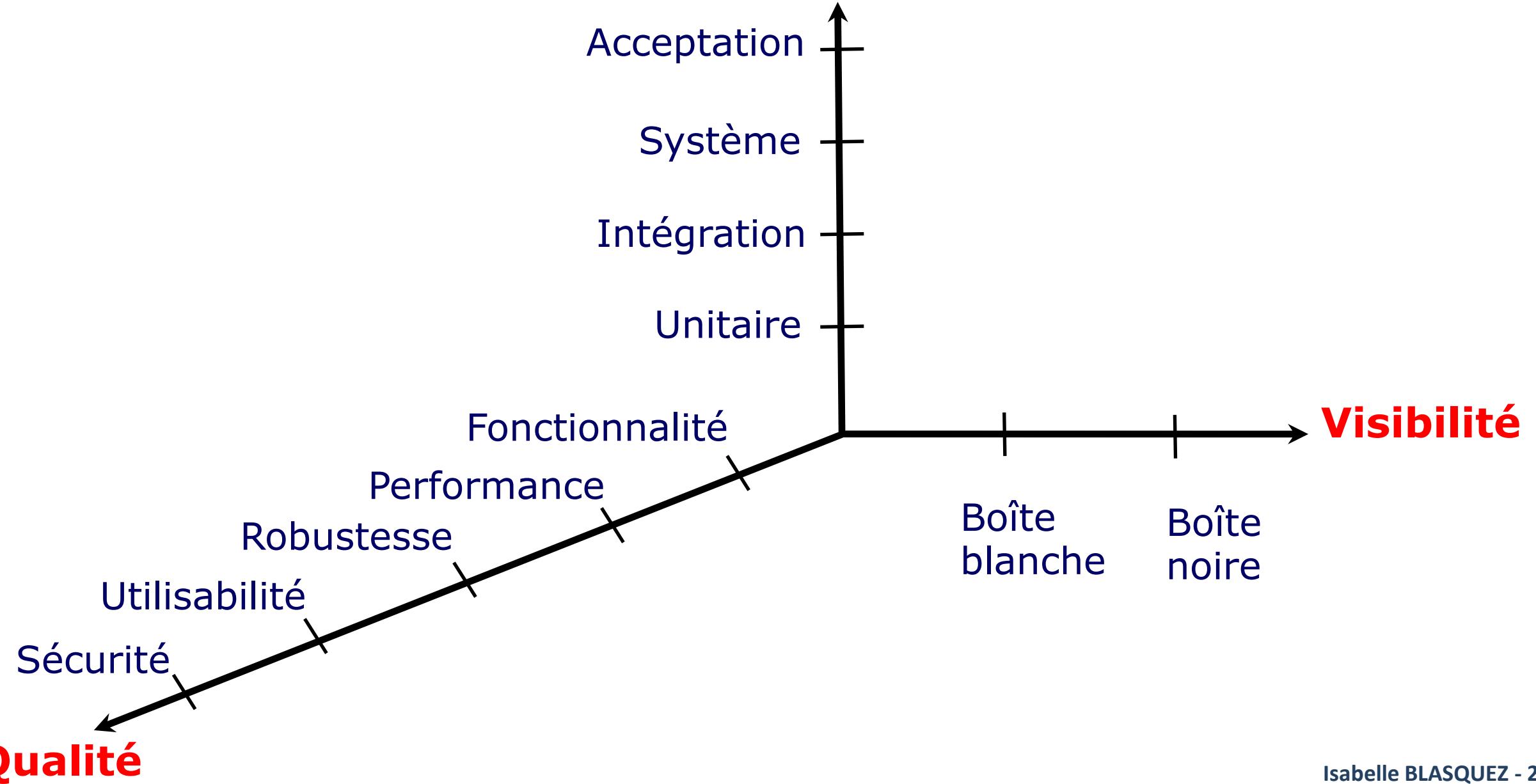
Conception

Implémentation



Typologie des tests logiciels classiques

Granularité



Un récapitulatif intéressant sur les tests front-end (avec des outils JS)

 OCTO Technology
@OCTOTechnology



Abonné

Nouvelle #refcard OCTO pour réussir vos
#tests #frontend. Make life simpler !
[#architecture](http://buff.ly/1N7xGq2)

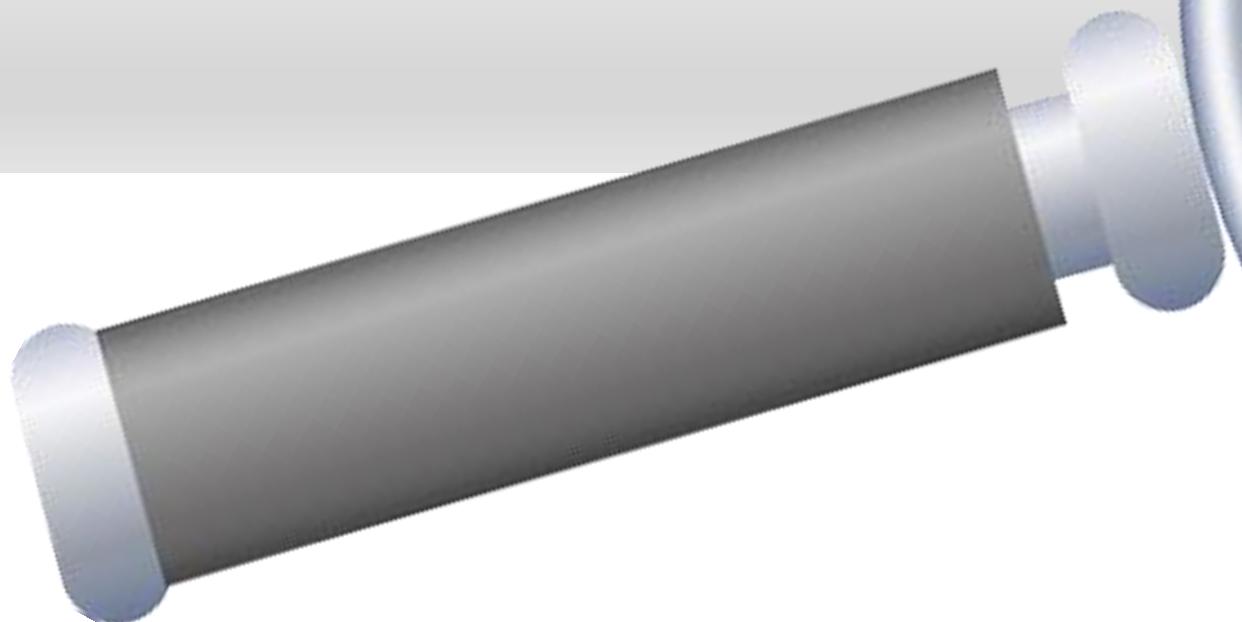


Extrait : <https://twitter.com/OCTOTechnology/status/649868083063062528/photo/1>

A consulter sur : <http://fr.slideshare.net/OCTOTechnology/test-sur-tous-les-fronts>

Le test dans un

**Développement
agile**



**XP est à l'origine
des processus de tests agiles ...**

Un test agile ?

Un test agile a les mêmes fonctionnalités qu'un test classique,

*Mais de par son côté **eXtreme**, il doit être*

Ecrit avant le code
*(approche test **FIRST**)*

Automatisé et isolé

Rejoué fréquemment
en toute confiance

Un code de test de qualité : *Clean Test*

FAST

INDEPENDANT

REPEATABLE

SELF-VALIDATING

TIMELY

Amélioration de la lisibilité du test : AAA

ARRANGE

```
@Test  
public void testShouldReturnHelloWorldWhenNothing()  
{  
    MaClasse monObjet = new MaClasse();
```

ACT

```
String resultat = monObjet.maMethodeATester();
```

ASSERT

```
Assert.assertEquals("Hello World!", resultat);  
}
```

Granularité des tests agiles

**Test Unitaire
(relatif au développeur)**

Tests techniques

**Test d'Acceptation
(relatif au client)**

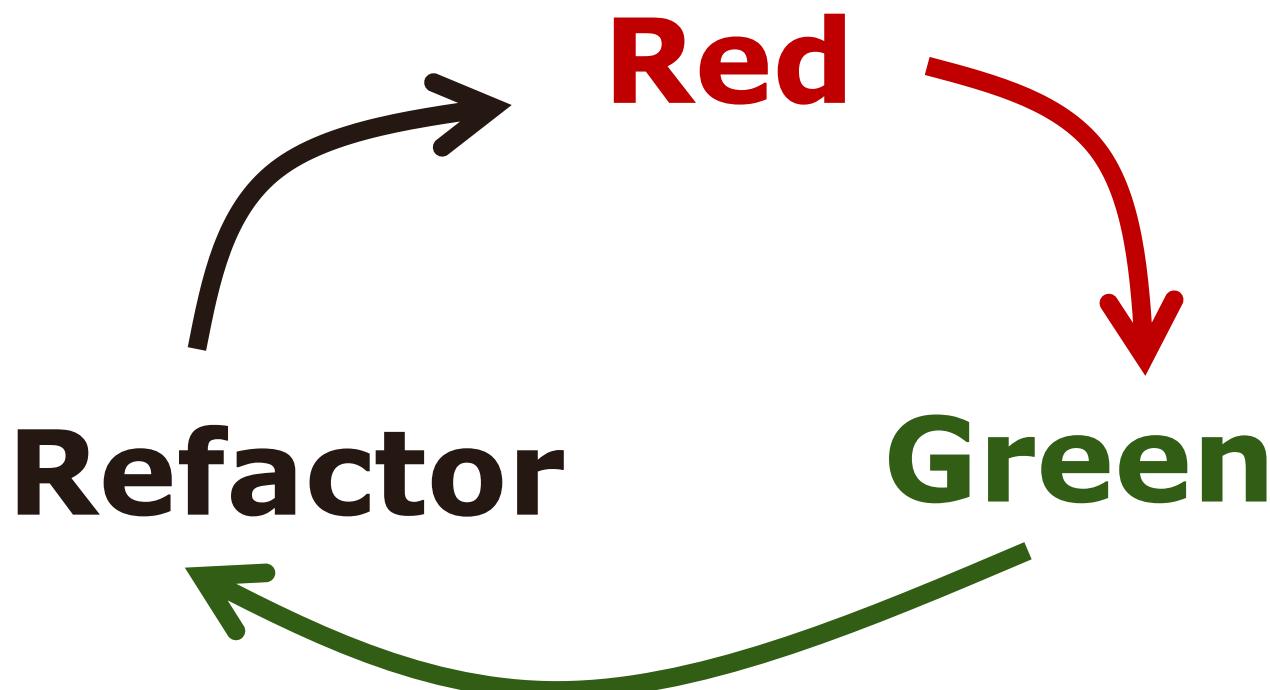
Tests métiers

Cycle de Développement Agile :

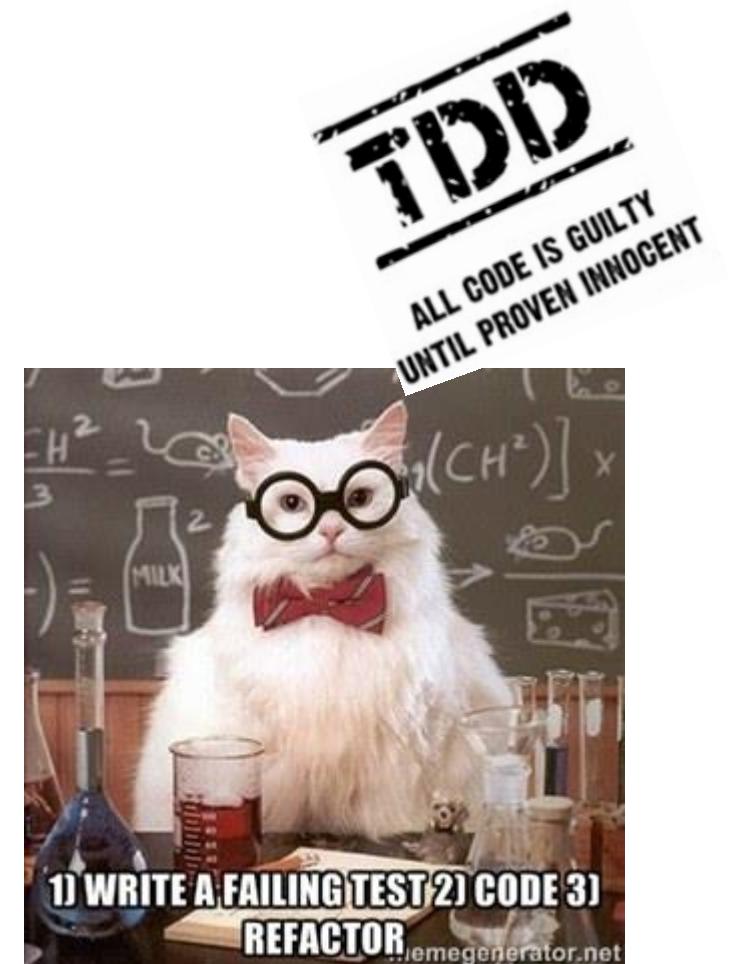
TDD

(Test Driven Développement)

Test Driven Développement



Itérations « baby-step » rapides
(de quelques secondes à quelques minutes)



Exemple Simple de TDD : Démarche

Code de tests

```
import static org.junit.Assert.*;  
import org.junit.Test;  
  
public class TestMaClasse {  
  
    @Test  
    public void testShouldReturnHelloWorldWhenNothing() {  
        MaClasse monObjet = new MaClasse();  
        String resultat = monObjet.maMethodeATester();  
        assertEquals("Hello World!", resultat);  
    }  
}
```

Pas de code sans test !

Un nouveau test =
un nouveau comportement

Code de Production

```
public class MaClasse {  
  
    public String maMethodeATester() {  
        return "Hello World!";  
    }  
}
```

Ecrire au plus vite un code
qui fait passer les tests

Les règles de simplicité (eXtreme Programming)

1. Tous les tests passent

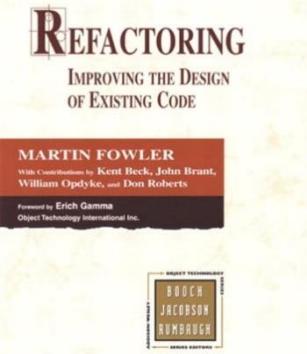
- * PASS ALL TESTS
- * CLEAR, EXPRESSIVE, & CONSISTENT
- * DUPLICATES NO BEHAVIOR OR CONFIGURATION
- * MINIMAL METHODS, CLASSES, & MODULES

3. DRY : pas de duplication
(Don't Repeat Yourself)

2. Révéler l'intention du code (lisibilité)

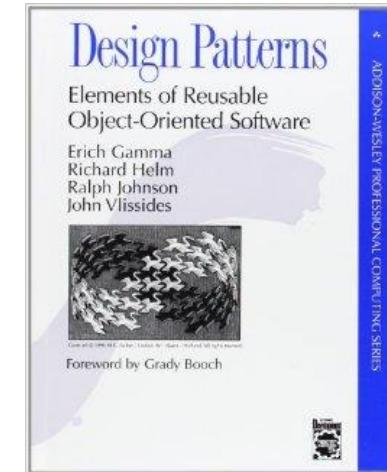
4. Le plus petit nombre d'éléments (complexité)

Le refactoring pour obtenir un code de qualité



***Un refactoring (remaniement) consiste à changer
la structure interne d'un logiciel
sans en changer son comportement observable
(M. Fowler)***

lisibilité
duplication
complexité



*Le refactoring contribue à travers tous les cycles du TDD
à l'obtention d'une **conception simple et évolutive**
appelée **conception émergente***

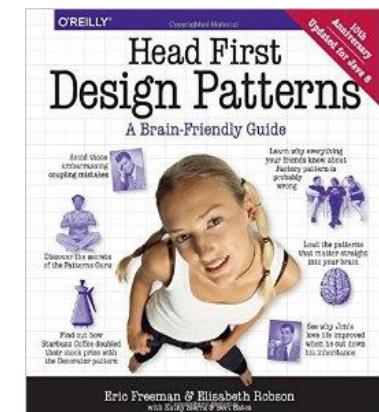
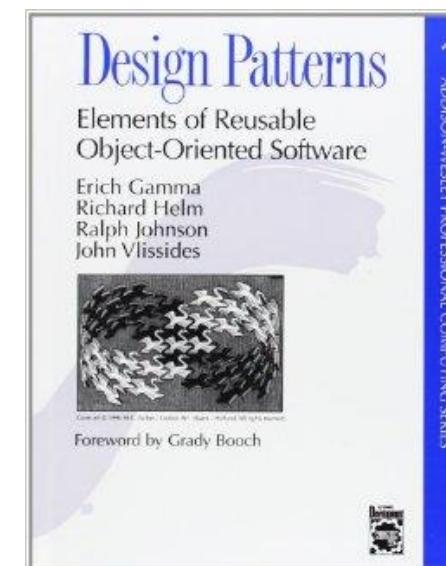
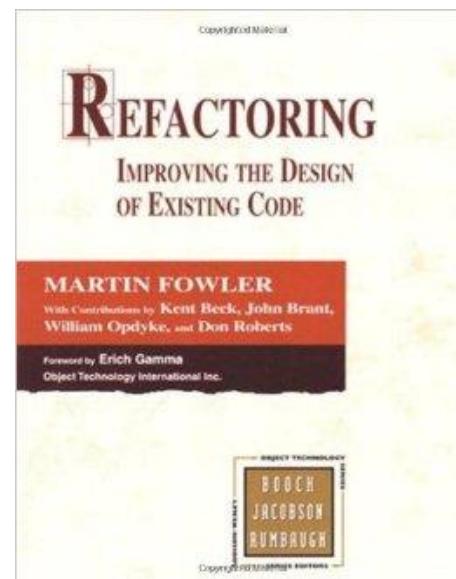
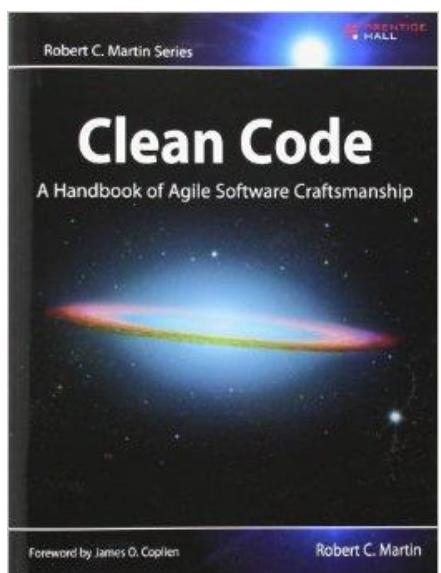
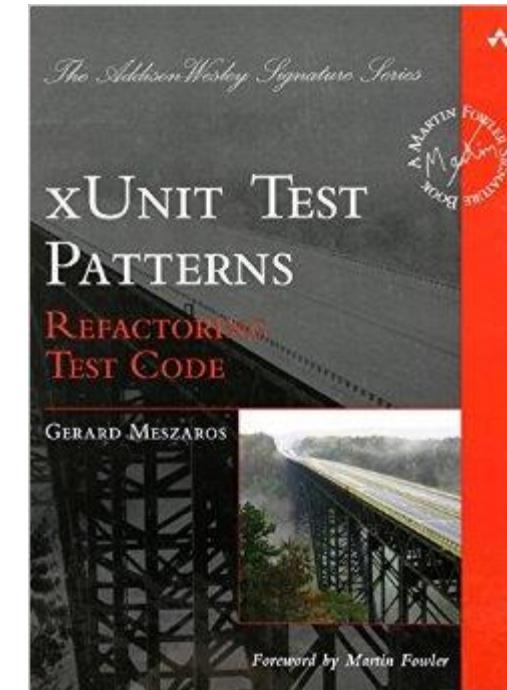
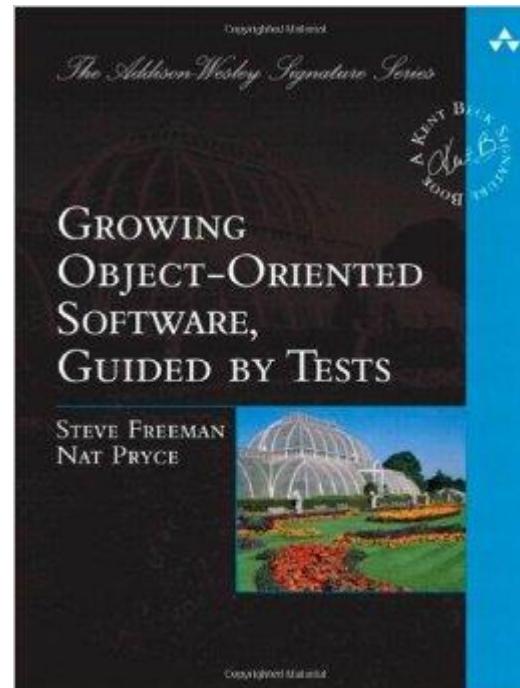
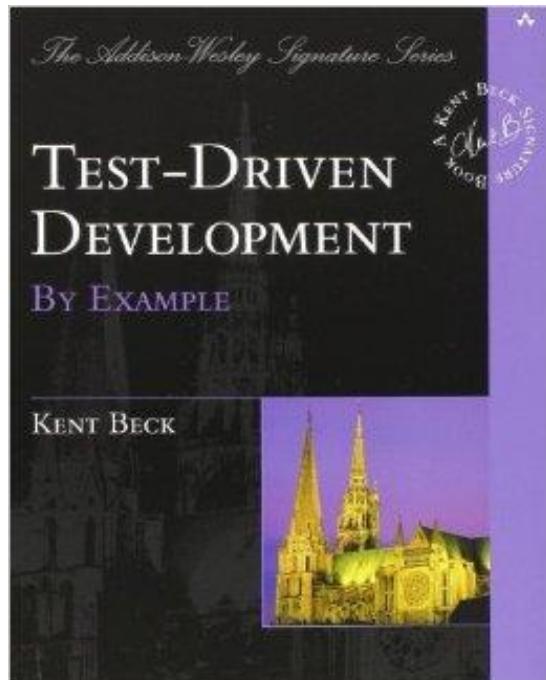
TDD : une technique de conception ?

*The act of writing a unit test
is more an act of design than of verification.
It is also more an act of documentation than of verification
(Robert. C. Martin)*

Test Driven

Developpement
Design

TDD : une discipline de programmation ...



Cycle de Développement Agile :

ATDD

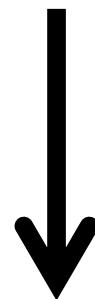
(Acceptance Test

Driven Développement)

Acceptance Test Driven Développement

Ateliers de
Spécification
(exemples)

Discuss → **Distill**



Demo ← **Develop**

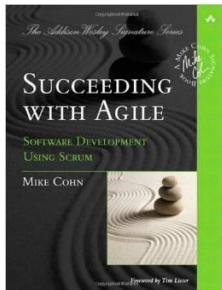
Right Product ?
Validation du comportement attendu
& Tests Exploratoires

TDD or not ?

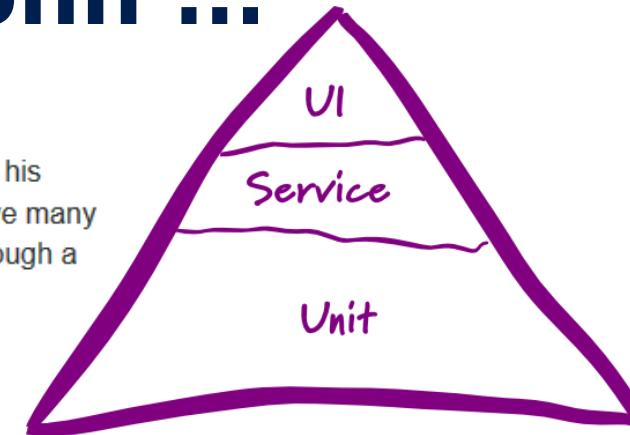


Les tests dans un développement Agile

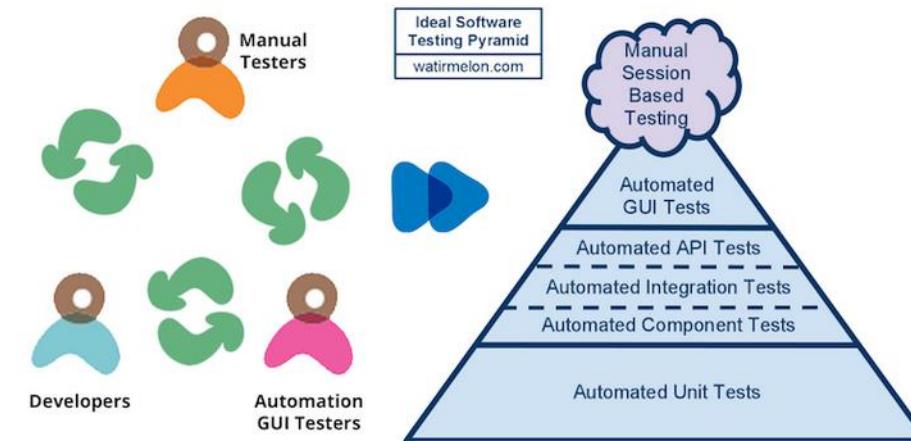
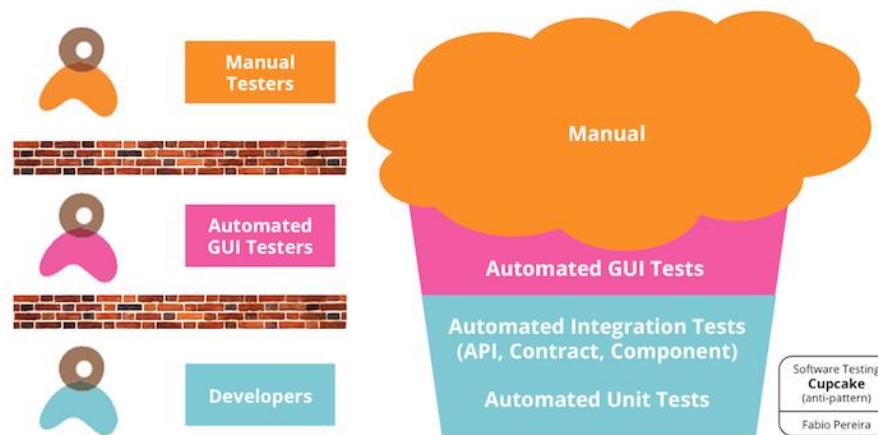
Des Tests oui, mais pas n'importe comment : Pyramide des tests de Mike Cohn ...



The test pyramid is a concept developed by [Mike Cohn](#), described in his book [Succeeding with Agile](#). Its essential point is that you should have many more low-level unit tests than high level end-to-end tests running through a GUI.



Introducing the Cupcake Anti-Pattern for software testing:



Extraits :

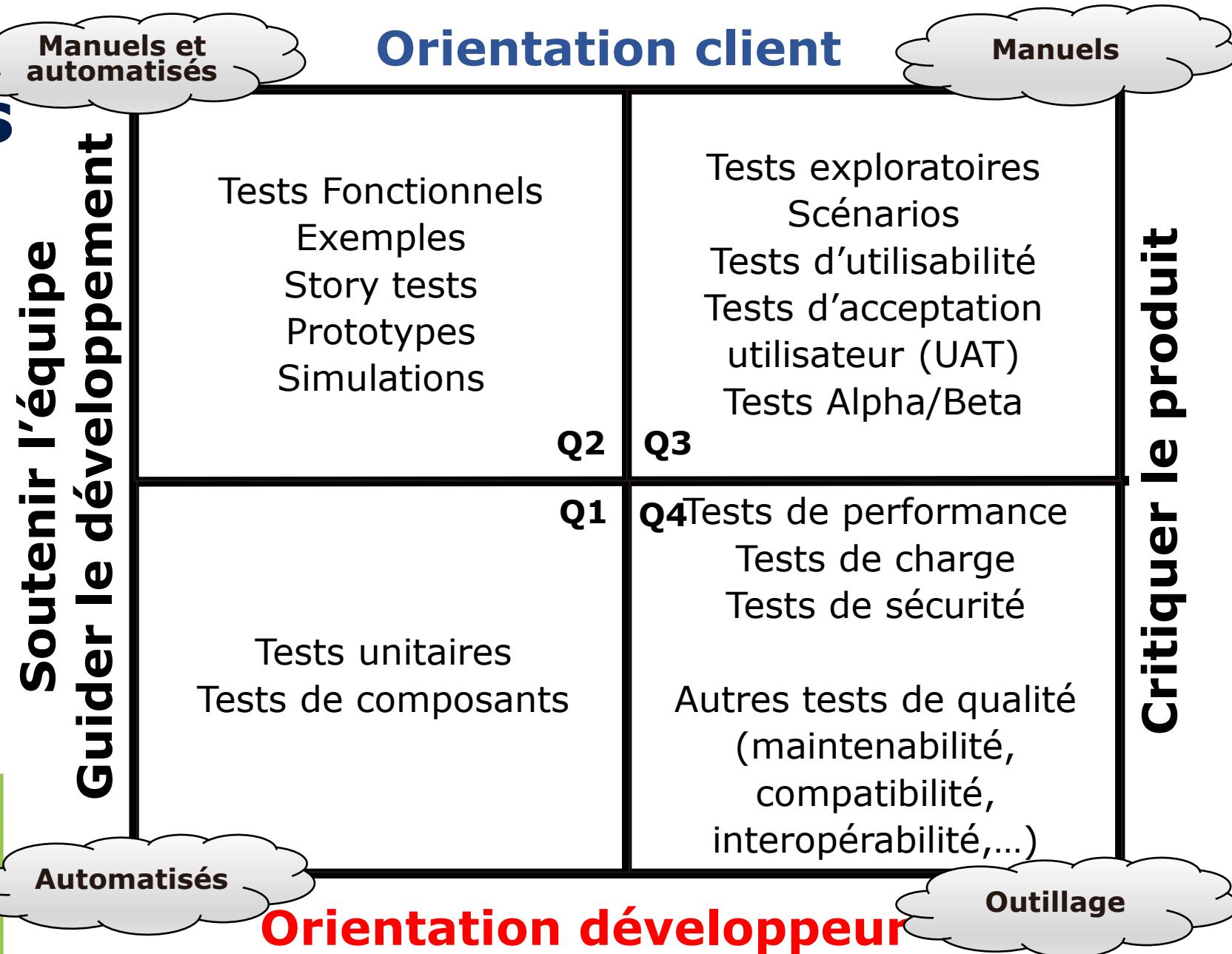
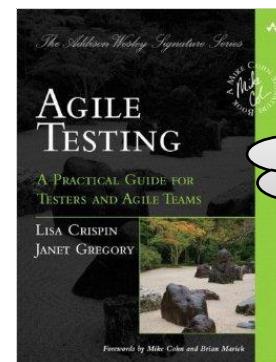
<http://www.thoughtworks.com/insights/blog/introducing-software-testing-cupcake-anti-pattern>

<http://martinfowler.com/bliki/TestPyramid.html>

A lire (Blog Mike Cohn) : <http://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid>

Comment les tests sont-ils utilisés dans le développement agile ?

Quadrant Agile



Voir : <http://lisacrispin.com/agile-testing-book-is-now-out/>
<http://lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants/>

Impact des tests agiles ...

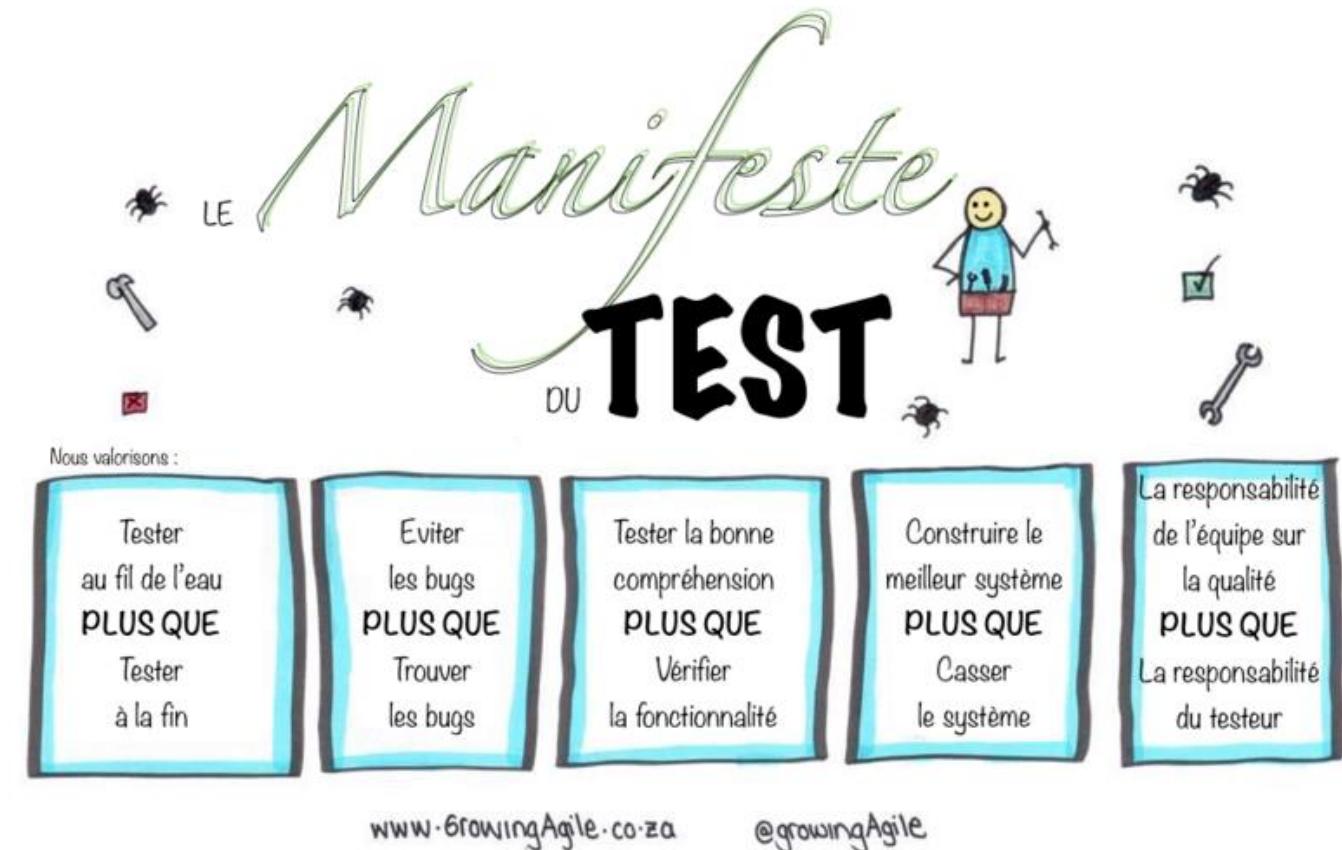


Nouveaux principes du test agile

Extrait d'Agile Testing (Livre blanc Valtech)

À télécharger sur :

<http://www.valtech.fr/fr/downloadfile/8251/2691>



Extrait : <http://wiki.ayeba.fr/Le+Manifeste+du+Test>

Isabelle BLASQUEZ - 2016

**Pas d'Agilité sans
excellence technique !**

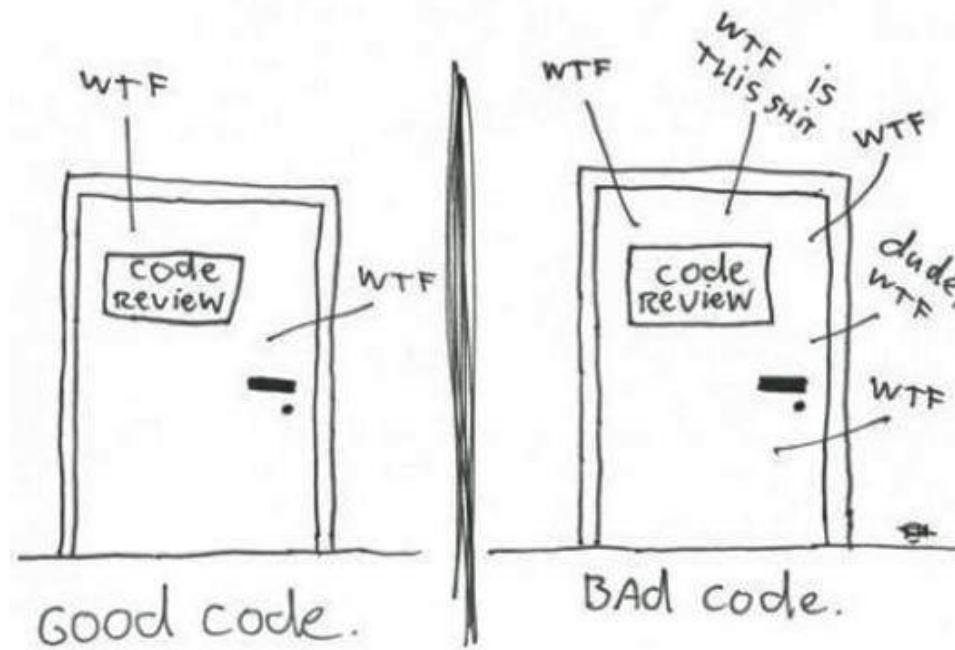
Pas d'agilité sans excellence technique !

9ème principe agile

Une attention continue à l'excellence technique et
à une bonne conception renforce l'Agilité.

Bien sûr, vous écrivez toujours un code de qualité

The ONLY VALID MEASUREMENT
OF CODE QUALITY: WTFs/MINUTE



(c) 2008 Focus Shift/OSNews/Thom Holwerda - <http://www.osnews.com/comics>

**Bien sûr, vous écrivez toujours un code de qualité,
facile à comprendre, à étendre et à modifier ...**

- En nommant correctement votre code pour bien montrer son intention,**
- En limitant les code smells grâce au refactoring ...**
- ... afin de faire émerger une conception simple**

Quand vous programmez, vous dépensez plus de temps à lire du code qu'à en écrire

Vous manipulez les blocs de codes source de la même manière qu'un sculpteur le fait pour des blocs d'argile.

Donc un langage qui génère un code source désagréable à lire est aussi horripilant pour un programmeur que l'est une terre grumeleuse pour un sculpteur.



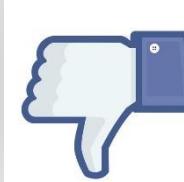
Paul Graham

(investisseur capital-risque et développeur Lisp)

Bien sûr, vous écrivez toujours un code très lisible (clair et expressif) ...

c'est un tri de tableau
(qui représente quoi ?)

```
List<int[]> theList = new ArrayList<>();  
  
public List<int[]> getFlg() {  
    List<int[]> list1 = new ArrayList<int[]>();  
    for (int[] x : theList)  
        if (x[0] == 4)  
            list1.add(x);  
    return list1;  
}
```



... en fonction de la valeur
stockée en index 0
(qui représente quoi ?)

... qui doit être égale à 4
(pourquoi ?)

et qui renvoie quoi ?

C'est pas mieux là ?...

```
List<Cell> gameBoard = new ArrayList<Cell>();  
  
public List<Cell> getFlaggedCell() {  
    List<Cell> flaggedCells = new ArrayList<Cell>();  
    for (Cell cell : gameBoard)  
        if (cell.isFlagged())  
            flaggedCells.add(cell);  
    return flaggedCells;  
}
```



... un code qui révèle son intention !

```
List<int[]> theList = new ArrayList<>();

public List<int[]> getFlg() {
    List<int[]> list1 = new ArrayList<int[]>();
    for (int[] x : theList)
        if (x[0] == 4)
            list1.add(x);
    return list1;
}
```

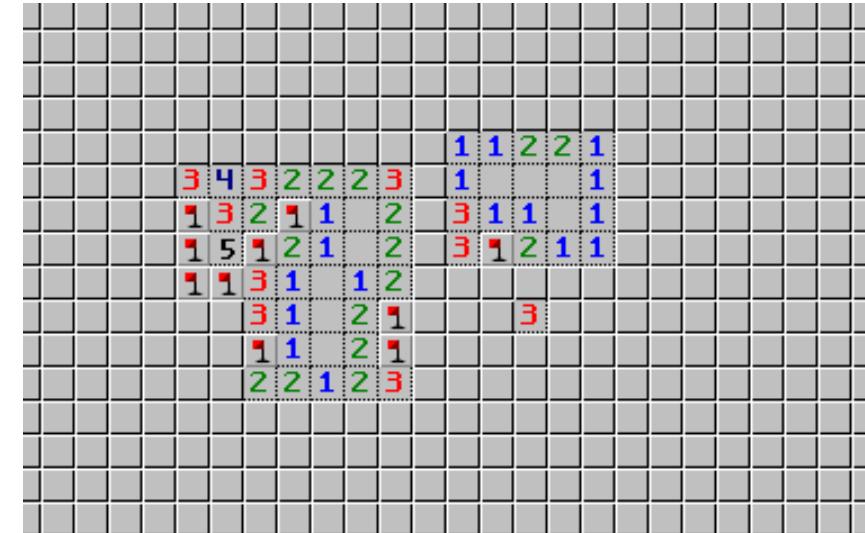


Un nommage explicite et un partage des responsabilités permet de contextualiser le code et de révéler clairement son intention

Renommer les variables

ALT+SHIFT+R (Eclipse)

getFlg() → getFlaggedCell()
theList → gameBoard
x → cell
list1 → flaggedCells



```
List<Cell> gameBoard = new ArrayList<Cell>();

public List<Cell> getFlaggedCell() {
    List<Cell> flaggedCells = new ArrayList<Cell>();
    for (Cell cell : gameBoard)
        if (cell.isFlagged())
            flaggedCells.add(cell);
    return flaggedCells;
}
```

Faire apparaître une nouvelle classe responsable de la cellule

Classe Cell et méthode isFlagged

... Avec des commentaires toujours très pertinents

COMMENTS IN REAL LIFE

```
/*
 * A comment to please checkstyle
 */

/*
 * Set the port
 *
 * @params port
 */
public void setPort(PORT port) {this.port=port}

...
    } /* end for */
    dao.flush();
    default :
        break;
    } /* end switch */
} /* end if */
} /* end if */
} catch ...
```

Extrait : http://avernois.github.io/prez-clean_code/#/5/1

Isabelle BLASQUEZ - 2016

Pertinence des commentaires ...

Over Commented Code



Extrait de : <https://twitter.com/CiaranMcNulty/status/517654863623487488>

Nécessité du commentaire ?



Extrait de : <https://twitter.com/nzkoz/status/538892801941848064>

Code Comments



Extrait de : <https://twitter.com/moh/status/659476321999818752>

Pertinence du commentaire :
Eclaircissement ou confusion ?

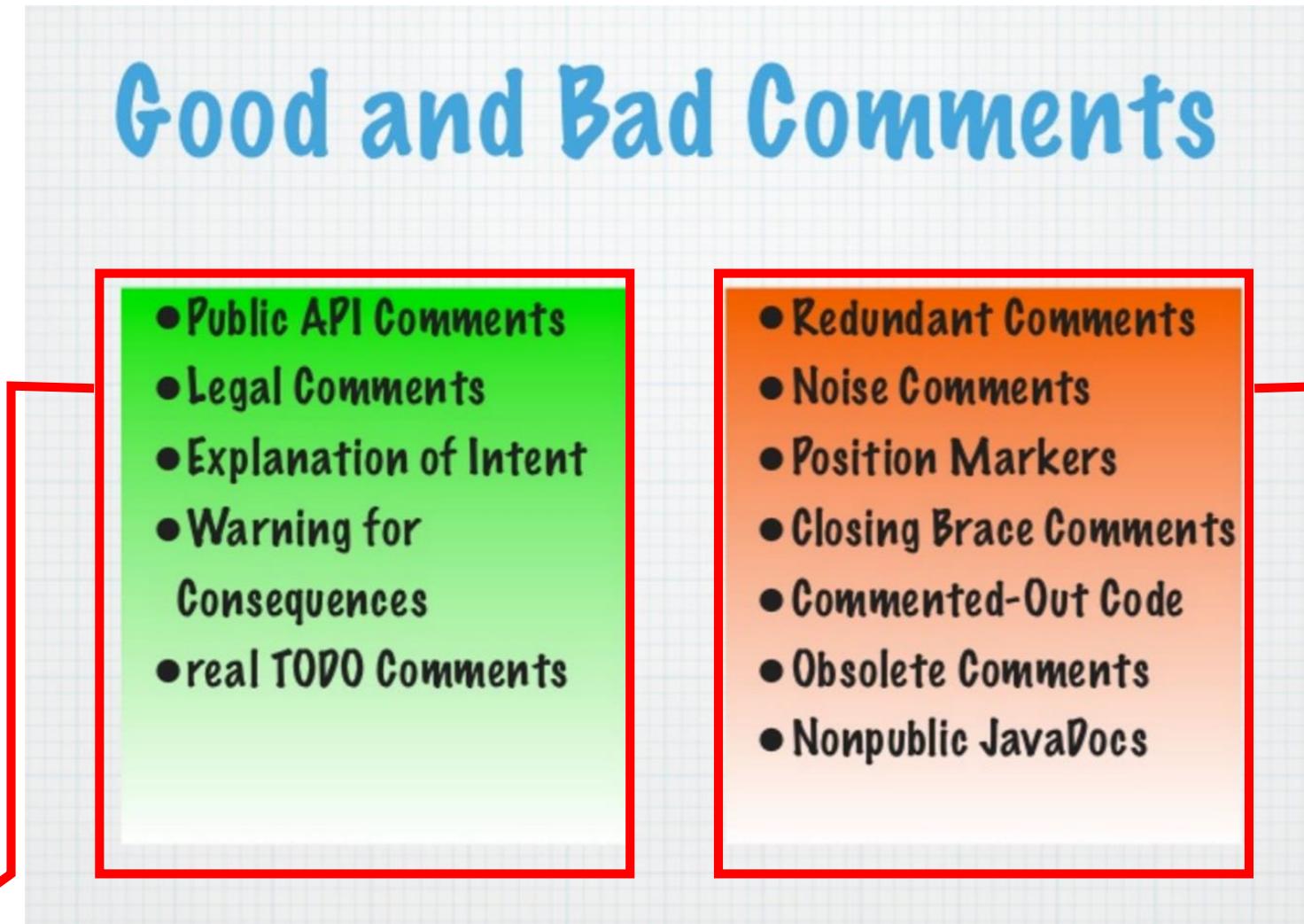
Comments are always failure

Uncle Bob

Don't comment bad code. Rewrite it.

Brian W. Kernighan, P.J. Plaugher

mais parfois avec des commentaires nécessaires ...

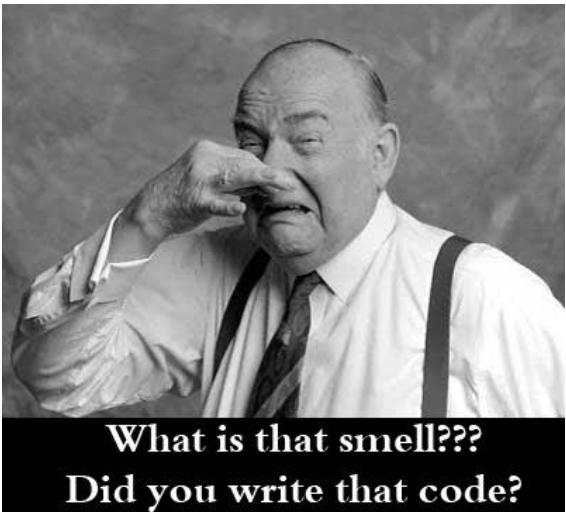


Pourquoi le code fait ça (WHY)

Ce que fait le code (HOW)

**TESTS TELLS ME WHAT
CODE TELLS ME HOW
COMMENT, IF NEEDED, TELLS ME WHY**

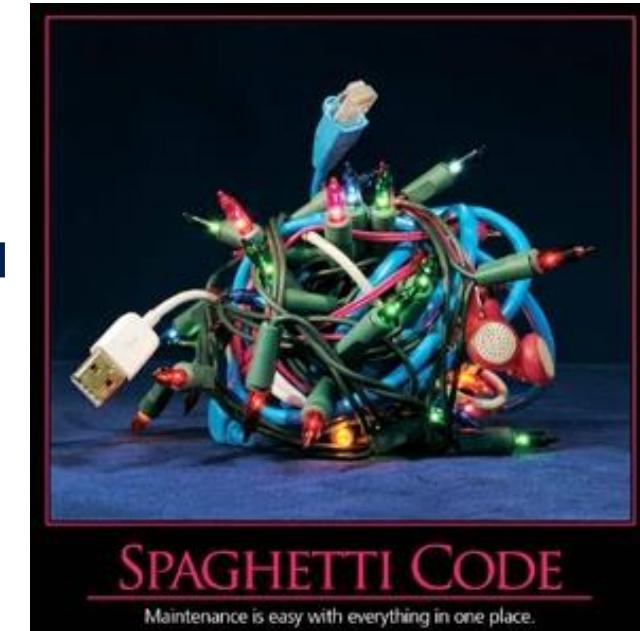
Et quand le code n'est pas *clean*



... il *smell*



et donne bien souvent du

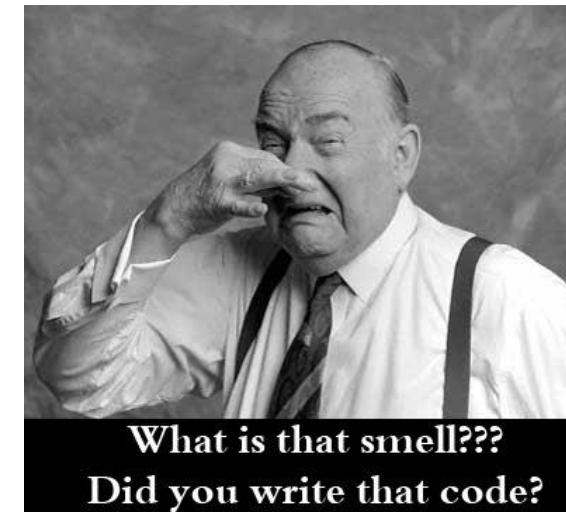
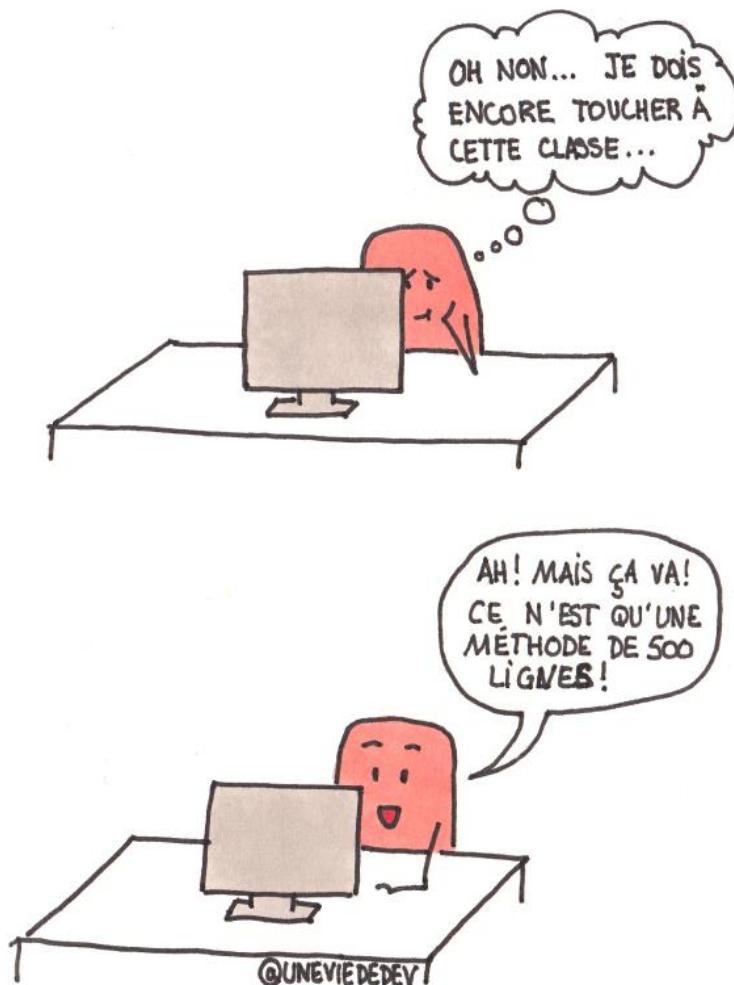


Images extraits :

<http://de.slideshare.net/iksgmbh/clean-codevortraggearconf>

<http://www.rogoit.de/webdesign-typo3-blog-duisburg/clean-code-regeln-smells-und-heuristiken/>

Une méthode bien trop longue ⇒ un code smell ...



Exemple de code smells ...

Code Smells Within Classes

Comments

There's a fine line between comments that illuminate and comments that obscure. Are the comments necessary? Do they explain "why" and not "what"? Can you refactor the code so the comments aren't required? And remember, you're writing comments for people, not machines.

Long Method

All other things being equal, a shorter method is easier to read, easier to understand, and easier to troubleshoot. Refactor long methods into smaller methods if you can.

Long Parameter List

The more parameters a method has, the more complex it is. Limit the number of parameters you need in a given method or use an object to combine the parameters.

Duplicated code

Duplicated code is the bane of software development. Stamp out duplication whenever possible. You should always be on the lookout for more subtle cases of near-duplication, too.
Don't Repeat Yourself!

Conditional Complexity Watch out for large conditional logic blocks, particularly blocks that tend to grow larger or change significantly over time. Consider alternative object-oriented approaches such as decorator, strategy, or state.

Combinatorial Explosion

You have lots of code that does *almost* the same thing.. but with tiny variations in data or behavior. This can be difficult to refactor-- perhaps using generics or an interpreter?

Large Class

Large classes, like long methods, are difficult to read, understand, and troubleshoot. Does the class contain too many responsibilities? Can the large class be restructured or broken into smaller classes?

Code Smells Between Classes

Alternative Classes with Different Interfaces

If two classes are similar on the inside, but different on the outside, perhaps they can be modified to share a common interface.

Primitive Obsession

Don't use a gaggle of primitive data type variables as a poor man's substitute for a class. If your data type is sufficiently complex, write a class to represent it.

Data Class

Avoid classes that passively store data. Classes should contain data *and* methods to operate on that data, too.

Data Clumps

If you always see the same data hanging around together, maybe it belongs together. Consider rolling the related data up into a larger class.

Refused Bequest

If you inherit from a class, but never use any of the inherited functionality, should you really be using inheritance?

Inappropriate Intimacy

Watch out for classes that spend too much time together, or classes that interface in inappropriate ways. Classes should know as little as possible about each other.

Indecent Exposure

Beware of classes that unnecessarily expose their internals. Aggressively refactor classes to minimize their public surface. You should have a compelling reason for every item you make public. If you don't, hide it.

Feature Envy

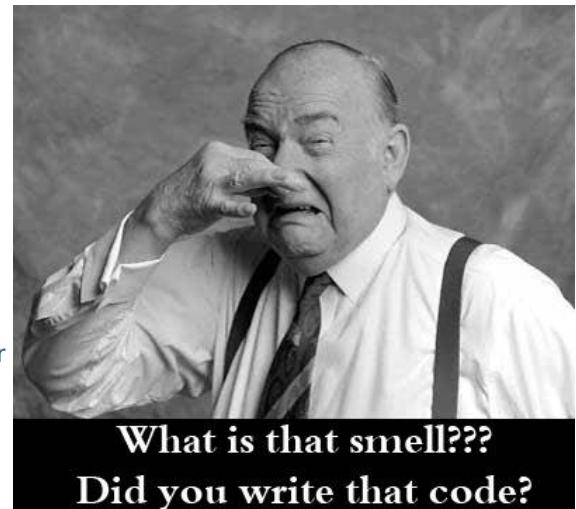
Methods that make extensive use of another class may belong in another class. Consider moving this method to the class it is so envious of.

Lazy Class

Classes should pull their weight. Every additional class increases the complexity of a project. If you have a class that isn't doing enough to pay for itself, can it be collapsed or combined into another class?

Message Chains

Watch out for long sequences of method calls or temporary variables to get routine data. Intermediaries are dependencies in disguise.



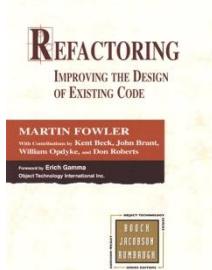
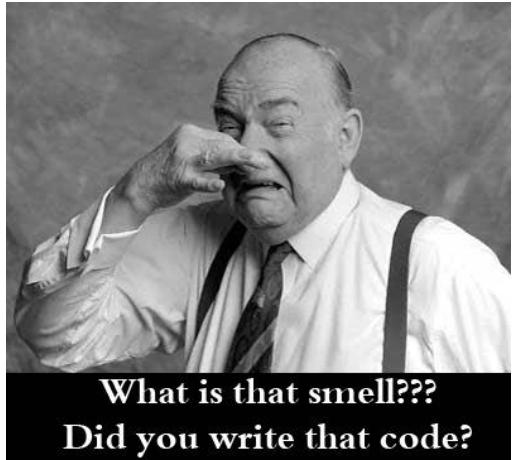
Liste non exhaustive extraite de : <http://blog.codinghorror.com/code-smells/>
où vous trouverez de nombreux autres smell codes ...

Voir aussi la Taxonomy des codes Smell : <http://badcodesmellstaxonomy.mikamantyla.eu/>

Image : <http://www.rogoit.de/webdesign-typo3-blog-duisburg/clean-code-regeln-smells-und-heuristiken/>

Isabelle BLASQUEZ - 2016

Un Code Smell : Que faire ?



Refactoring



Pas de refactoring
sans test !

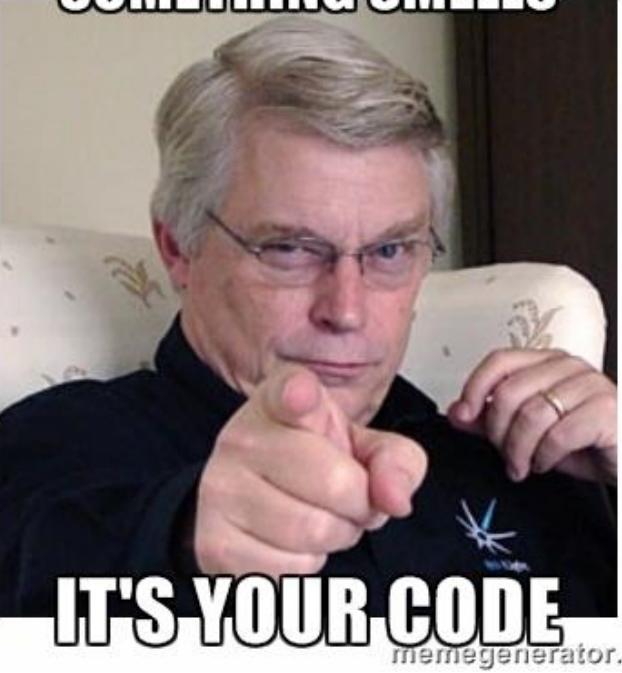


Extrait :

<http://www.passetoncode.fr/cours/feux-tricolore>

Isabelle BLASQUEZ - 2016

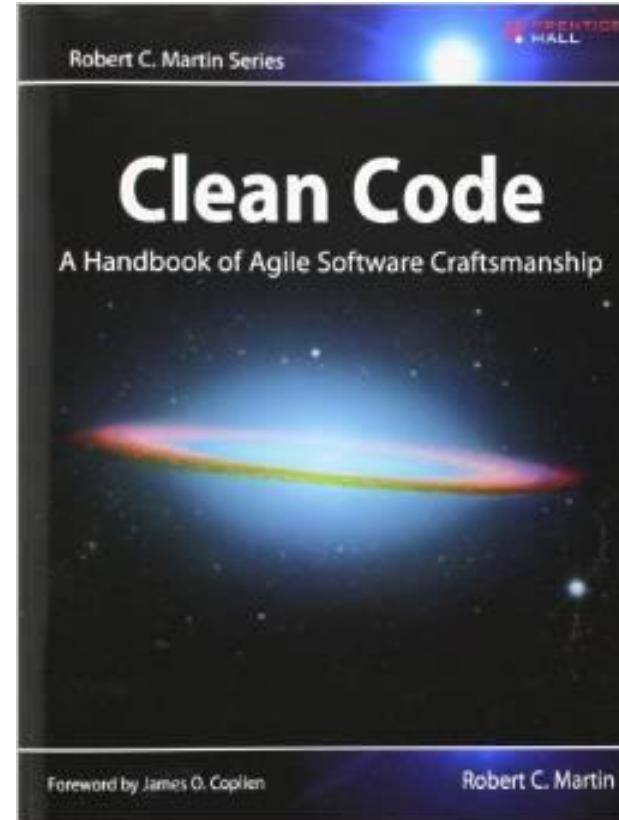
SOMETHING SMELLS



<http://memegenerator.net/instance/54386514>



<http://codesmells.net/>



<http://www.amazon.fr/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882> Isabelle BLASQUEZ - 2016

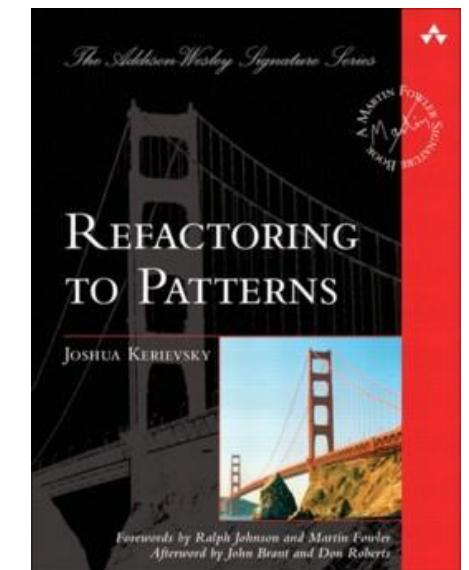
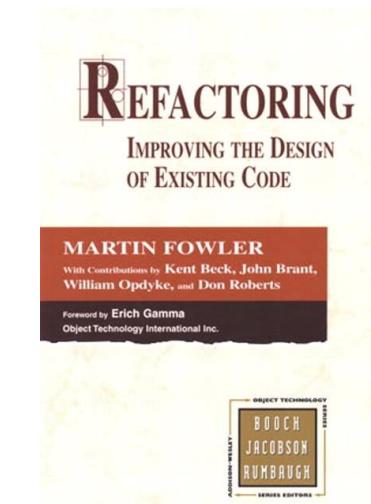
Aide mmoire : Smells to refactorings

Smell	Refactoring
Alternative Classes with Different Interfaces: occurs when the interfaces of two classes are different and yet the classes are quite similar. If you can find the similarities between the two classes, you can often refactor the classes to make them share a common interface. [F 85, K 43]	Unify Interfaces with Adapter [K 247] Rename Method [F 273] Move Method [F 142]
Combinatorial Explosion: A subtle form of duplication, this smell exists when numerous pieces of code do the same thing using different combinations of data or behavior. [K 45]	Replace Implicit Language with Interpreter [K 209]
Comments (a.k.a. Deodorant): When you feel like writing a comment, first try "to refactor so that the comment becomes superfluous" [F 87]	Rename Method [F 273] Extract Method [F 110] Introduce Assertion [F 267]
Conditional Complexity: Conditional logic is innocent in its infancy, when it's simple to understand and contained within a few lines of code. Unfortunately, it rarely ages well. You implement several new features and suddenly your conditional logic becomes complicated and expansive. [K 41]	Introduce Null Object [F 280, K 301] Move Embellishment to Decorator [K 144] Replace Conditional Logic with Strategy [K 129] Replace State-Altering Conditionals with State [K 166]
Data Class: Classes that have fields, getting and setting methods for the fields, and nothing else. Such classes are dumb data holders and are almost certainly being manipulated in far too much detail by other classes. [F 86]	Move Method [F 142] Encapsulate Field [F 206] Encapsulate Collection [F 208]
Data Clumps: Bunches of data that hang around together really ought to be made into their own object. A good test is to consider deleting one of the data values: if you did this, would the others make any sense? If they don't, it's a sure sign that you have an object that's dying to be born. [F 81]	Extract Class [F 149] Preserve Whole Object [F 288] Introduce Parameter Object [F 205]
Divergent Change: Occurs when one class is commonly changed in different ways for different reasons. Separating these divergent responsibilities decreases the chance that one change could affect another and lower maintenance costs. [F 79]	Extract Class [F 149]
Duplicated Code: Duplicated code is the most pervasive and pungent smell in software. It tends to be either explicit or subtle. Explicit duplication exists in identical code, while subtle duplication exists in structures or processing steps that are outwardly different, yet essentially the same. [F 76, K 39]	Chain Constructors [K 340] Extract Composite [K 214] Extract Method [F 110] Extract Class [F 149] Form Template Method [F 345, K 205] Introduce Null Object [F 280, K 301] Introduce Polymorphic Creation with Factory Method [K 88] Pull Up Method [F 322] Pull Up Field [F 320] Replace One/Many Distinctions with Composite [K 224] Substitute Algorithm [F 139] Unify Interfaces with Adapter [K 247]
Feature Envy: Data and behavior that acts on that data belong together. When a method makes too many calls to other classes to obtain data or functionality, Feature Envy is in the air. [F 80]	Extract Method [F 110] Move Method [F 142] Move Field [F 146]
Freeloader (a.k.a. Lazy Class): A class that isn't doing enough to pay for itself should be eliminated. [F 83, K 43]	Collapse Hierarchy [F 344] Inline Class [F 154] Inline Singleton [K 114]
Inappropriate Intimacy: Sometimes classes become far too intimate and spend too much time delving into each others' private parts. We may not be prudes when it comes to people, but we think our classes should follow strict, puritan rules. Over-intimate classes need to be broken up as lovers were in ancient days. [F 85]	Move Method [F 142] Move Field [F 146] Change Bidirectional Association to Unidirectional Association [F 20] Extract Class [F 149] Hide Delegate [F 157]



Extrait de : <http://www.industriallogic.com/wp-content/uploads/2005/09/smellstorefactorings.pdf>

Ralise  partir de :



Techniques de refactoring

Catalog of Refactorings



Martin Fowler

10 December 2013

This catalog of refactorings includes those refactorings described in my original book on Refactoring, together with the Ruby Edition.

Using the Catalog ▶

Tags

- associations
- encapsulation
- generic types
- interfaces
- class extraction
- GOF Patterns
- local variables
- vendor libraries
- errors
- type codes
- method calls
- organizing data
- inheritance
- conditionals
- moving features
- composing methods

- Add Parameter
- Change Bidirectional Association to Unidirectional
- Change Reference to Value
- Change Unidirectional Association to Bidirectional
- Change Value to Reference
- Collapse Hierarchy
- Consolidate Conditional Expression
- Consolidate Duplicate Conditional Fragments
- Decompose Conditional
- Duplicate Observed Data
-
- Pull Up Constructor Body
- Pull Up Field
- Pull Up Method
- Push Down Field
- Push Down Method
- Recompose Conditional
- Remove Assignments to Parameters
- Remove Control Flag
- Remove Middle Man
- Remove Named Parameter
- Remove Parameter
- Remove Setting Method

<http://refactoring.com/catalog/>

Consolidate Conditional Expression

You have a sequence of conditional tests with the same result.

Combine them into a single conditional expression and extract it.

```
double disabilityAmount() {  
    if (_seniority < 2) return 0;  
    if (_monthsDisabled > 12) return 0;  
    if (_isPartTime) return 0;  
    // compute the disability amount
```

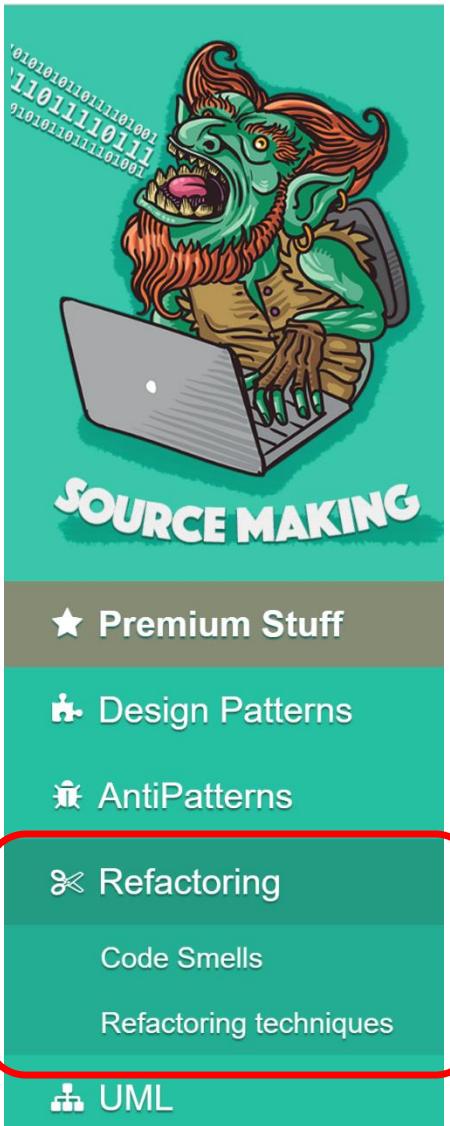


```
double disabilityAmount() {  
    if (isNotEligibleForDisability()) return 0;  
    // compute the disability amount
```



A découvrir
petit à petit ...

Jetez également un petit coup d'œil à ...



<https://sourcemaking.com/refactoring>

Extract Method

Problem

You have a code fragment that can be grouped together.

```
void printOwing() {  
    printBanner();  
  
    //print details  
    System.out.println("name: " + name);  
    System.out.println("amount: " + getOutstanding());  
}
```

Solution

Move this code to a separate new method (or function) and replace the old code with a call to the method.

```
void printOwing() {  
    printBanner();  
    printDetails(getOutstanding());  
}  
  
void printDetails(double outstanding) {  
    System.out.println("name: " + name);  
    System.out.println("amount: " + outstanding);  
}
```

Java C# PHP Python

Why Refactor

The more lines found in a method, the harder it is to figure out what the method does. This is the main reason for this refactoring.

Besides eliminating rough edges in your code, extracting methods is also a step in many other refactoring approaches.

A découvrir
petit à petit ...

Benefits

- More readable code! Be sure to give the new method a name that describes the method's purpose: `createOrder()`, `renderCustomerInfo()`, etc.
- Less code duplication. Often the code that is found in a method can be reused in other places in your program. So you can replace duplicates with calls to your new method.
- Isolates independent parts of code, meaning that errors are less likely (such as if the wrong variable is modified).

How to Refactor

1. Create a new method and name it in a way that makes its purpose self-evident.
2. Copy the relevant code fragment to your new method. Delete the fragment from its old location and put a call for the new method there instead.
Find all variables used in this code fragment. If they are declared inside the fragment and not used outside of it, simply leave them unchanged – they will become local variables for the new method.
3. If the variables are declared prior to the code that you are extracting, you will need to pass these variables to the parameters of your new method in order to use the values previously contained in them. Sometimes it is easier to get rid of these variables by resorting to [Replace Temp with Query](#).
4. If you see that a local variable changes in your extracted code in some way, this may mean that this changed value will be needed later in your main method. Double-check! And if this is indeed the case, return the value of this variable to the main method to keep everything functioning.

Sans refactoring, attention la dette technique vous guette

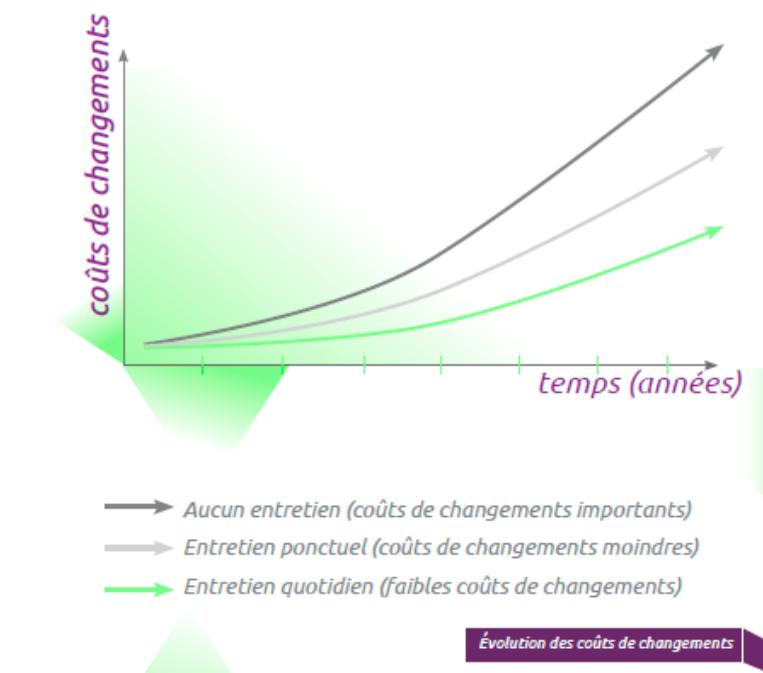
La dette technique est une *métaphore du développement logiciel* inventée par **Ward Cunningham**.

La définition de Wikipedia est éclairante:

Il s'inspire du concept existant de dette dans le contexte du financement des entreprises et l'applique au domaine du développement logiciel. En résumé, quand on code au plus vite et de manière non optimale, on contracte une dette technique que l'on rembourse tout au long de la vie du projet sous forme de temps de développement de plus en plus long et de bugs de plus en plus fréquents.

Just tried to explain technical debt to a customer, had to pull this out again...

Voir la traduction



Sources : techtrends.xebia.fr <http://blog.octo.com/maitriser-sa-dette-technique/> <https://twitter.com/khellang/status/626716128379830273>

A lire : <http://www.occitech.fr/blog/2014/11/intro-dette-technique/> (Traduction de Technical Debt101 de Maiz Lulkin)

<http://c2.com/cgi/wiki?WardExplainsDebtMetaphor> et <http://promyze.com/wp-content/uploads/2016/06/LaDetteTechnique.pdf> (la dette technique expliquée !)

A visionner : <https://www.youtube.com/watch?v=d3XYhlIikeA>

Clean Code : Bonnes pratiques



Quelles bonnes pratiques ?

SOLID

YAGNI

DRY

SoC

GRASP

KISS

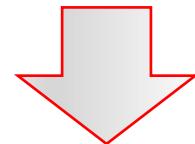
Loi de Demeter

- * PASS ALL TESTS
- * CLEAR, EXPRESSIVE, & CONSISTENT
- * DUPLICATES NO BEHAVIOR OR CONFIGURATION
- * MINIMAL METHODS, CLASSES, & MODULES

From STUPID to SOLID

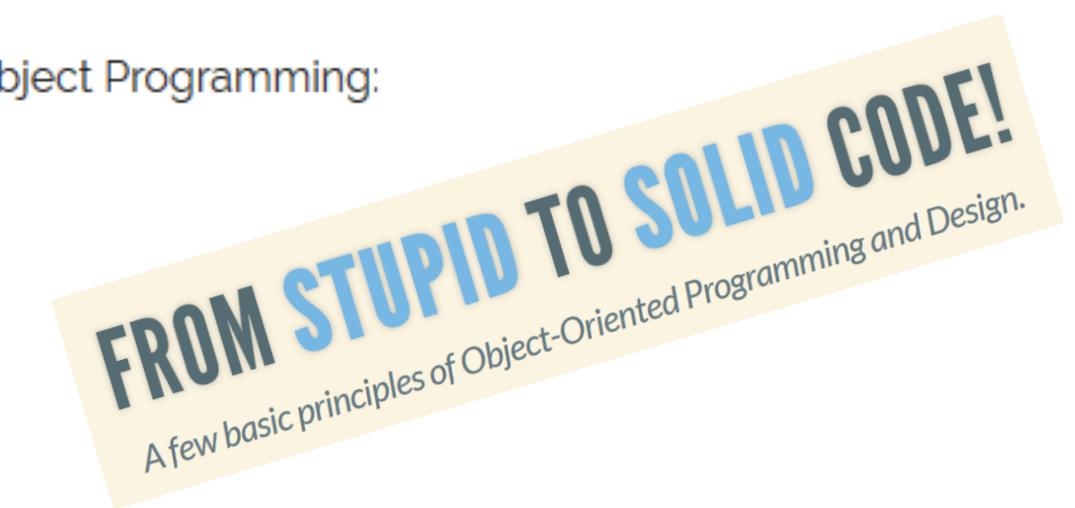
STUPID is an acronym that describes bad practices in Oriented Object Programming:

- Singleton
- Tight Coupling
- Untestability
- Premature Optimization
- Indescriptive Naming
- Duplication



SOLID is an acronym that stands for **five basic principles** of Object-Oriented Programming and design to *fix* STUPID code:

- Single Responsibility Principle
- Open/Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle



SOLID

Software Development is not a Jenga game



Les principes SOLID : SRP

***Une classe doit avoir
une et une seule responsabilité***

How many responsibilities?

```
class Employee {  
    public Pay calculatePay() {...}  
    public void save() {...}  
    public String describeEmployee() {...}  
}
```

3 responsabilités
⇒ 3 classes à écrire !



SINGLE RESPONSIBILITY PRINCIPLE

Ce n'est pas parce qu'on peut le faire
qu'il faut le faire



Images extraits de : <http://lostechies.com/derickbailey/2009/02/11/solid-development-principles-in-motivational-pictures/>

En savoir plus : http://lostechies.com/wp-content/uploads/2011/03/pablos_solid_ebook.pdf

Extrait : <http://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/>

<http://thetechnologistinyou.blogspot.fr/2012/09/solid-principles-quick-introduction.html>

Isabelle BLASQUEZ - 2016

Les principes SOLID : OCP



OPEN CLOSED PRINCIPLE

Une opération à cœur ouvert n'est pas nécessaire lorsqu'on enfile un vêtement

Wrong
Class ABC{
FindAreaOfSquare()
FindAreaOfTriangle()
}

Right
Abstract Class Shape
{
FindArea()
}

Une classe doit être ouverte à l'extension, mais fermée à la modification.
(possibilité d'étendre le comportement sans modifier la classe)



Open Closed Principle

You don't need to rewire your MoBo to plug in "Mr Happy"

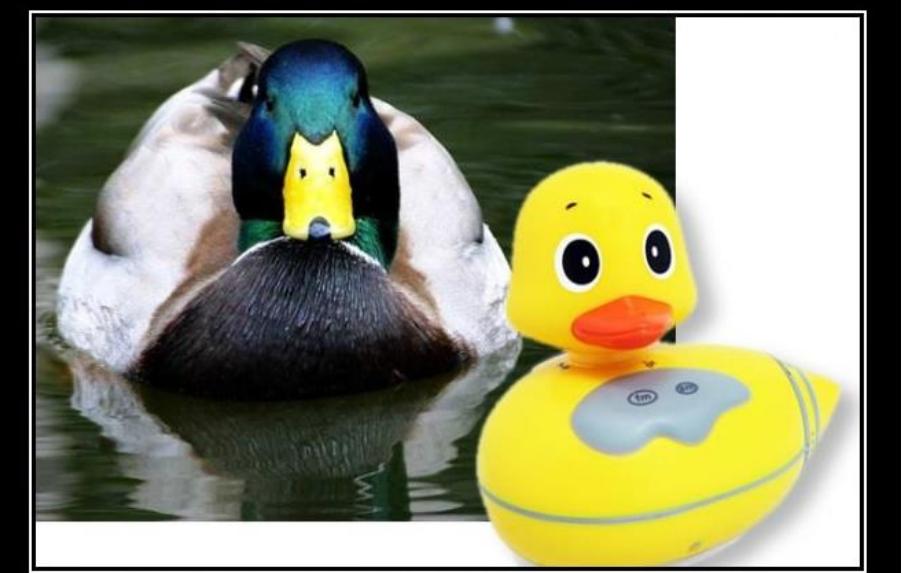
Images extraites de : <http://lostechies.com/derickbailey/2009/02/11/solid-development-principles-in-motivational-pictures/>

En savoir plus : http://lostechies.com/wp-content/uploads/2011/03/pablos_solid_ebook.pdf

Extrait : <http://zereturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/>

<http://thetechnologistinyou.blogspot.fr/2012/09/solid-principles-quick-introduction.html>

Les principes SOLID : LSP



LISKOV SUBSTITUTION PRINCIPLE

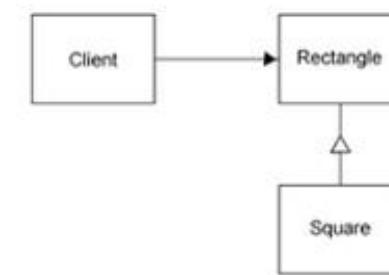
Ca ressemble à un canard,
ça cancane comme un canard, mais à besoin de piles.
Vous avez sûrement la mauvaise abstraction.

*Toutes les classes filles doivent fonctionner
de la même manière que la classe de base
c-a-d avoir un comportement conforme
à la classe de base*



Une classe doit pouvoir être remplacée par une instance d'un de ses sous-types, sans modifier la cohérence du programme.

"Derived classes should extend without replacing the functionality of old classes"



Class Square derived from Rectangle violates LSP because we cannot use square class to create rectangle. Substitutability fails.

RECTANGLE - int length, int width (2 properties)
SQUARE -- int length (1 property)

Images extraits de : <http://lostechies.com/derickbailey/2009/02/11/solid-development-principles-in-motivational-pictures/>

En savoir plus : http://lostechies.com/wp-content/uploads/2011/03/pablos_solid_ebook.pdf

Extrait : <http://zereturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/>

<http://thetechnologistinyou.blogspot.fr/2012/09/solid-principles-quick-introduction.html>

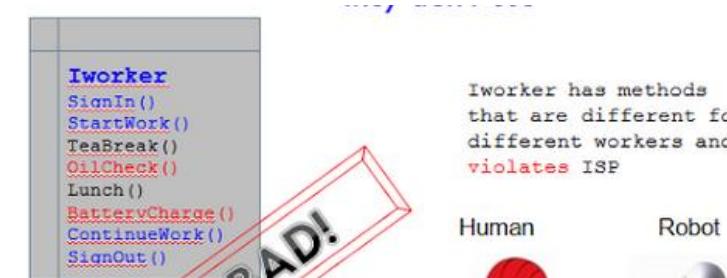
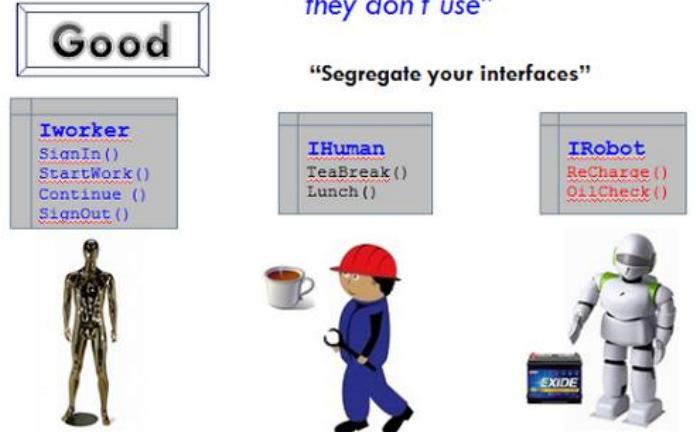
Les principes SOLID : ISP



INTERFACE SEGREGATION PRINCIPLE

Où voulez-vous brancher cela ?

"Clients should not be forced to depend upon interfaces that they don't use"



Interface Segregation Principle

If IRequireFood, I want to Eat(Food food) not, LightCandelabra() or LayoutCutlery(CutleryLayout preferredLayout)

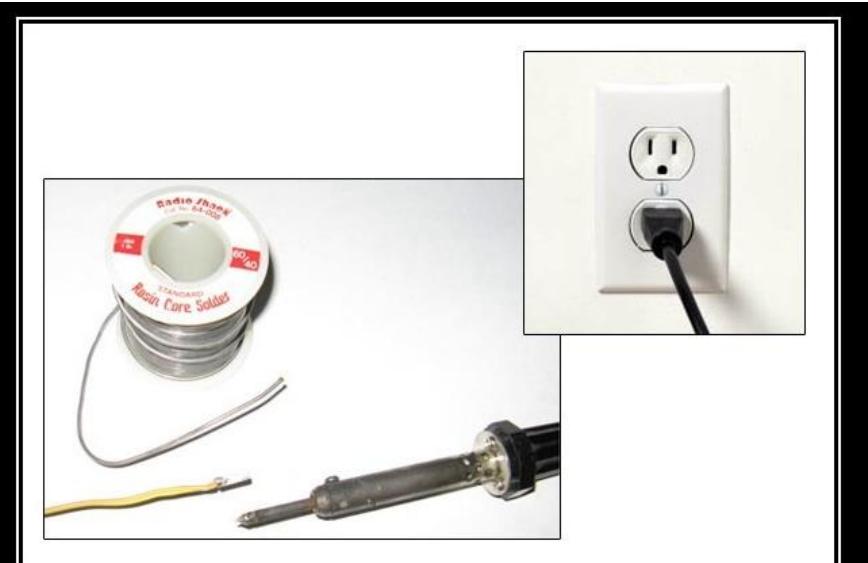
Images extraits de : <http://lostechies.com/derickbailey/2009/02/11/solid-development-principles-in-motivational-pictures/>

En savoir plus : http://lostechies.com/wp-content/uploads/2011/03/pablos_solid_ebook.pdf

Extrait : <http://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/>

<http://thetechnologistinyou.blogspot.fr/2012/09/solid-principles-quick-introduction.html>

Les principes SOLID : DIP



DEPENDENCY INVERSION PRINCIPLE

Est-ce que vous souderiez directement un branchement électrique dans un mur ?



Les modules de haut niveau ne doivent pas dépendre de modules de plus bas niveau.
Les deux doivent dépendre d'abstractions.

"Higher modules should not depend on lower modules. In such cases invert the dependency"



Wrong Dependency

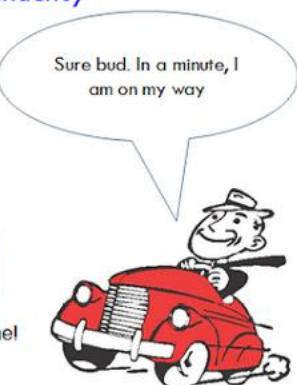


Be there in a minute.
Gettin' ready!
(hahaha)

"Higher modules should not depend on lower modules. In such cases invert the dependency"



Dependency Inverted
You need me, you call me!



Sure bud. In a minute, I am on my way

Images extraits de : <http://lostechies.com/derickbailey/2009/02/11/solid-development-principles-in-motivational-pictures/>

En savoir plus : http://lostechies.com/wp-content/uploads/2011/03/pablos_solid_ebook.pdf

Extrait : <http://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/>

<http://thetechnologistinyou.blogspot.fr/2012/09/solid-principles-quick-introduction.html>

A voir : <http://www.dotnetdeja.com/concevoir-des-applications-solid-avec-dip/> et <http://www.dotnetdeja.com/conception-applications-solid/>



SOLID

Software Development is not a Jenga game



SINGLE RESPONSIBILITY PRINCIPLE

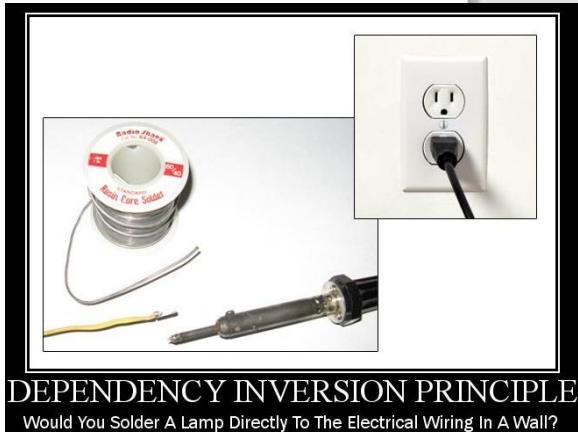
Just Because You Can, Doesn't Mean You Should



OPEN CLOSED PRINCIPLE

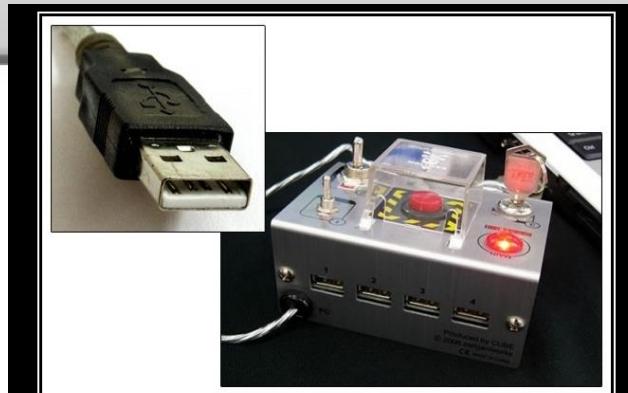
Open Chest Surgery Is Not Needed When Putting On A Coat

S . O . L . I . D



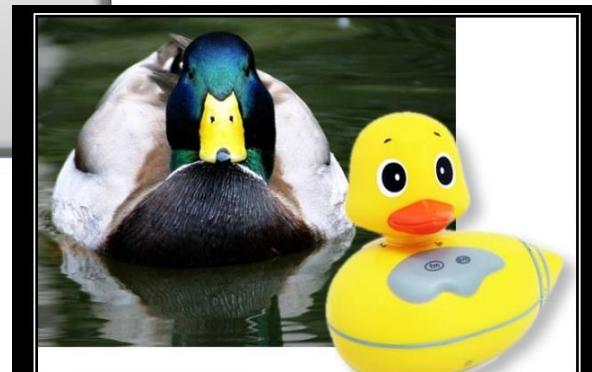
DEPENDENCY INVERSION PRINCIPLE

Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?



INTERFACE SEGREGATION PRINCIPLE

You Want Me To Plug This In, Where?

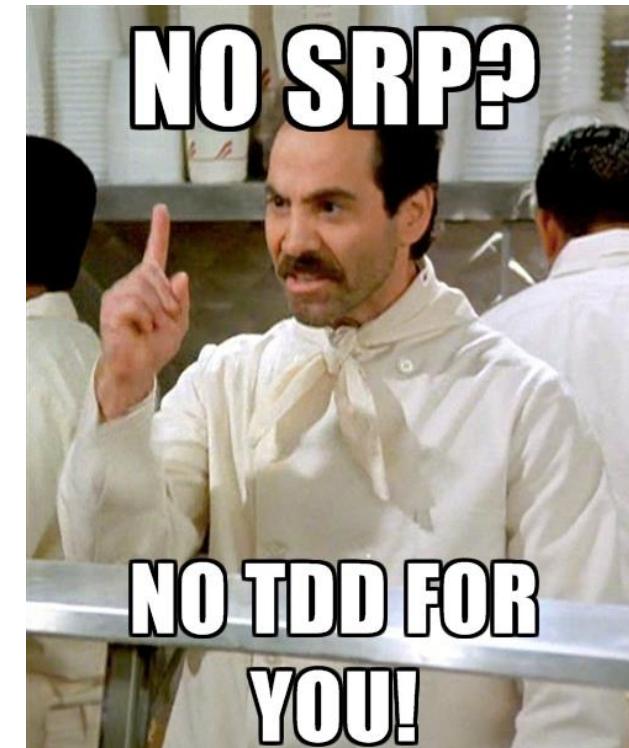
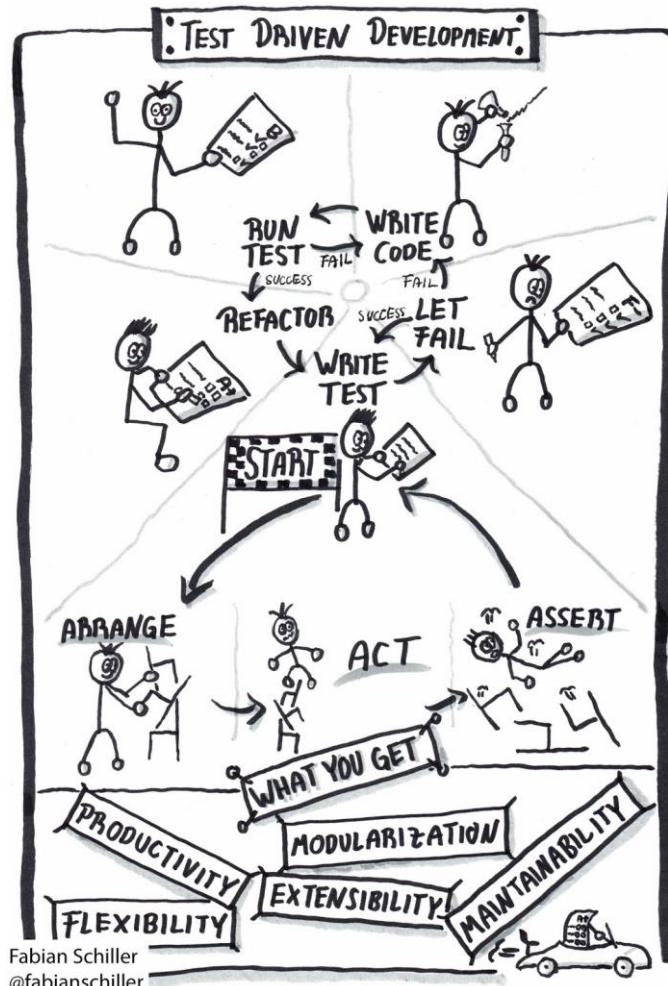


LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

Et bien sûr : Le Clean code au cœur du TDD

But du TDD d'après Ron Jeffries
« Clean code that works »



Le clean code et les objets calisthenics ...

OBJECT CALISTHENICS

Jeff Bay, in The ThoughtWorks Anthology, lists 9 rules to writing better Object Oriented code.

MOTIVATION

Readable Code, Comprehensible, Testable, Maintainable.

Juste pour information

OBJECT CALISTHENICS

Your code sucks, let's fix it!

CALISTHENICS

Term derived from greek, exercise, under the context of gymnastics.

RECAP'

1. Only One Level Of Indentation Per Method
2. Don't Use The ELSE Keyword
3. Wrap All Primitives And Strings
4. First Class Collections
5. One Dot Per Line
6. Don't Abbreviate
7. Keep All Entities Small
8. No Classes With More Than Two Instance Variables
9. No Getters/Setters/Properties

Extrait : <http://williamdurand.fr/2013/06/03/object-calisthenics/> (Très bons exemples illustrés en lien avec les principes SOLID ...)

En savoir plus : <http://www.cs.helsinki.fi/u/luontola/tdd-2009/ext/ObjectCalisthenics.pdf>

<http://fr.slideshare.net/KLabCscorpions-TechBlog/object-calisthenics-34557136>

<http://blog.netapsys.fr/devoxx-france-2014-les-object-calisthenics/>

Isabelle BLASQUEZ - 2016

Software Craftsmanship

**L'artisan, c'est celui qui met les mains dans sa matière première,
qui la sculpte, qui la modèle, qui la transforme, qui l'agence en
choisisissant à chaque fois l'outil adapté à ses besoins.**

Bref, son job est concret, pragmatique, il demande de la réflexion.

Extrait : <http://www.ithaquecoaching.com/articles/job-crafting-artisan-plaisir-travail-7744.html>



***A l'image du potier
qui travaille son argile en continu,
le développeur travaille son code
en permanence. Benoit Gantaume***

Extraits : <http://www.frenchweb.fr/craftsman-lartisan-du-code/221948>

Le Manifeste du Software Craftsmanship

Manifeste for Software Craftsmanship

Elever le niveau

En tant qu'aspirants Artisans du Logiciel, nous relevons le niveau du développement professionnel de logiciels par la pratique et en aidant les autres à acquérir le savoir-faire. Grâce à ce travail, nous avons appris à apprécier :

Pas seulement des logiciels opérationnels,
mais aussi **des logiciels bien conçus.**

Pas seulement l'adaptation aux changements,
mais aussi **l'ajout constant de la valeur.**

Pas seulement les individus et leurs interactions,
mais aussi **une communauté de professionnels.**

Pas seulement la collaboration avec les clients,
mais aussi **des partenariats productifs.**

C'est à dire qu'en recherchant les éléments de gauche, nous avons trouvé que les éléments de droite sont indispensables.

© 2009, les sous-signés.
Cette déclaration peut être copiée librement, quelle que soit la forme,
mais seulement dans sa totalité, y compris cette remarque.

Signer le Manifeste

Les mots d'ordre du Software Craftsmanship :

- **Raising the bar!**
- **Working code is not enough**
- **Doing the right thing, and do the thing right!**

En un mot : **professionnalisme.**

Il s'agit **d'attirer à nouveau l'attention sur les pratiques de code et d'ingénierie :**
TDD, Clean Code, refactoring, design, patterns...

Rééquilibrer la place du développeur dans une équipe de développement,
en le présentant comme un **artisan-développeur**.

Extrait : <http://www.touilleur-express.fr/2011/01/20/craftsmanship/>



Software Craftsmanship : Valeurs

Qualité

(conception simple, tests, TDD)

Humilité

(je me remets en question et je m'améliore en continu)

Pragmatisme

(je n'applique pas bêtement, je comprends ce que je fais et je m'adapte si nécessaire)

Professionalisme

(je traite mon client comme un partenaire, je sais dire non quand nécessaire)

Partage

(je partage mes acquis avec mes pairs, je m'enrichis chemin faisant)

Concret

(pas de certifications fumeuses, mais un retour aux **techniques fondamentales : OO, SOLID, DDD**)

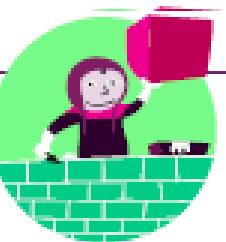
Software Craftsmanship : Principes

Take away
Craftsmanship
Trends



INITIER

- Définir les objectifs et aligner les visions de tous les participants du projet sur la notion de qualité logicielle :
 - Répondre aux besoins des utilisateurs,
 - Développer avec des coûts et des délais maîtrisés,
 - Présenter des qualités intrinsèques au projet : modularité, stabilité, évolutivité, lisibilité.
- Constituer et entretenir des équipes de craftsmen passionnés, impliqués, pragmatiques, humbles et légitimes.



CONSTRUIRE

- Favoriser la communication et les pratiques de pairing afin de maintenir l'homogénéité de l'équipe.
- Laisser les artisans choisir leurs outils, dans la mesure du possible et dans le respect des contraintes de l'entreprise.
- Encourager un environnement propice à la réalisation de tests, au maintien et au nettoyage (refactoring) fréquent du code.



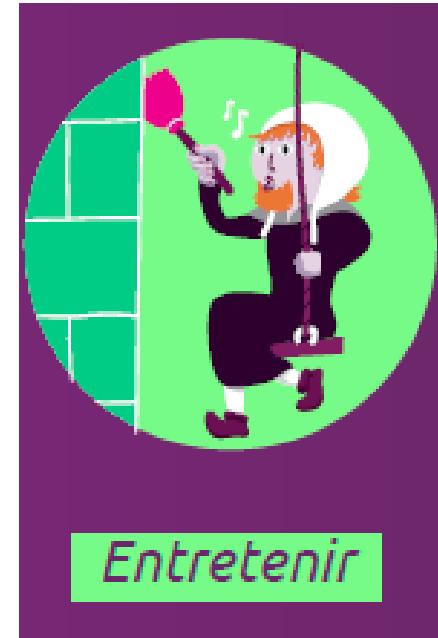
ENTREtenir

- Aménager des moments d'échanges techniques et méthodologiques à des horaires raisonnables, pour que chaque membre de l'équipe puisse s'enrichir du savoir de l'autre.
- Miser sur la communauté pour ne pas rester en marge des innovations.

Software Craftsmanship : action !



Software Craftsmanship Paris
Software Craftsmanship Toulouse
Software Craftsmanship Lyon
Software Craftsmanship Limoges ?



GitHub



<https://github.com/dojo-toulouse>

Google

<https://groups.google.com/forum/#!forum/software-craftsmanship-toulouse>

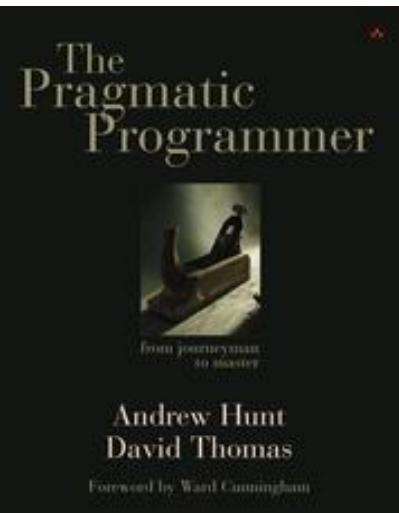


Des activités suivant les envies de chacun :

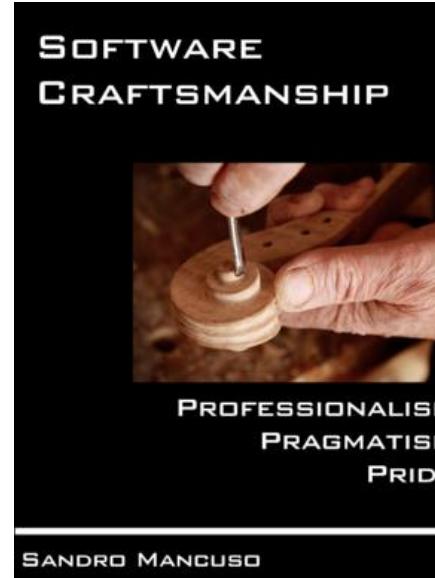
Code & Coffee, Coding Dojo, Code Retreat, Club de lecture, BYOC,
Code & Music, Atelier découverte thématique, présentation, ...

Isabelle BLASQUEZ - 2016

En savoir plus sur le Software Craftsmanship



[https://pragprog.com/book/tpp/
the-pragmatic-programmer](https://pragprog.com/book/tpp/the-pragmatic-programmer)



<https://leanpub.com/socra>



techtrends.xebia.fr



<http://www.octo.com/fr/livres-blancs> Isabelle BLASQUEZ - 2016

A méditer ...



Gilles Roustan

Artisan développeur Web



» Blog

» A propos de moi

Artisan développeur

12/06/2016

Avant-propos

Cet article est une retranscription de la conférence que j'ai donnée lors de l'agile tour de Montpellier le 13 octobre 2015. Pour mieux suivre, je vous conseille de regarder en parallèle les [slides de la conférence](#).

A propos de moi

Je m'appelle Gilles et je suis artisan développeur.

Dans cet article, je vais vous parler de moi, de mon métier et de ma famille.

Plus particulièrement de l'évolution de la perception de mon métier et comment le métier de mon père et de mon grand-père m'ont aidé à devenir un développeur plus heureux.

La mode du software craftsmanship

Il n'y a pas si longtemps, j'ai changé de poste et quand j'ai fait mon CV, j'ai mis que j'étais **artisan développeur**.

Il y a quelques années, j'ai vu débarquer cette *mode* lors de [conférences](#), dans des articles sur Software Craftsmanship.

J'ai trouvé ça sympa comme titre : artisan développeur ! Un contraste entre du moderne et du traditionnel. Et moi comme un mouton, j'ai trouvé ça cool et je me suis dépêché de modifier mon profil twitter.

How Can You Say You're A Software Crafts(wo)man?

Or what makes you a Software Crafts(wo)man

This is a question that came up yesterday during our latest meetup with our local [French Riviera Software Craftsmanship Community](#) (note: a very nice round table session on the beach of Juan Les Pins ☺).

This post is my answer to this question.

What Software Craftsmanship is not

It's not an elitist club of people who think to hold the truth of how to write the perfect code. This is an important aspect because it is related to how much inclusive or exclusive a group of people is (and also to how much nice people are). I've been following the international Software Craftsmanship movement since some time now and I met some of them during [SoCraTes UK](#) and I learned how much important are things like kindness, mentorship and empathy for a Software Crafts(wo)man. Software development is not only techniques and programming: it's also relationship and communication with your peers.

It's not a certification of one's capabilities to write good software (assuming it does exist really). A person yesterday was doubtful whether he could call himself a *software craftsman* or not because he is not working in TDD or applying all the time the SOLID principles.

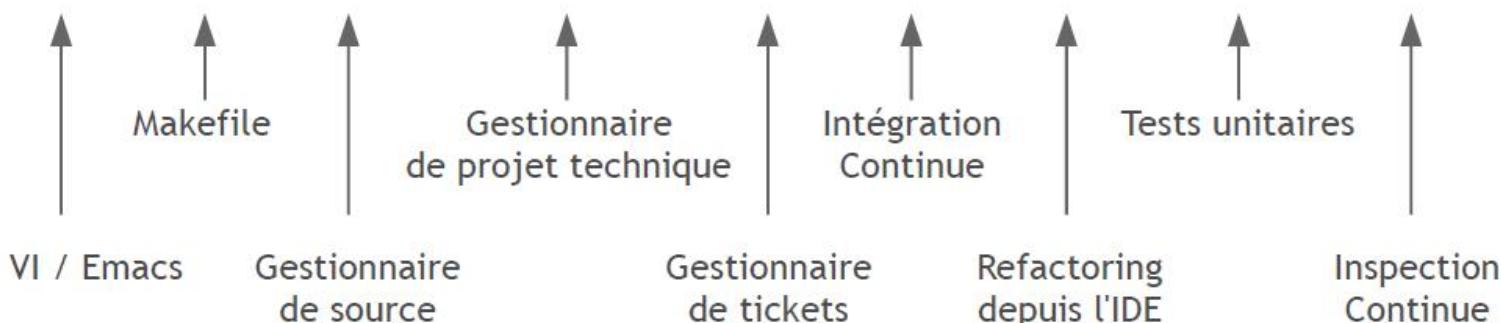
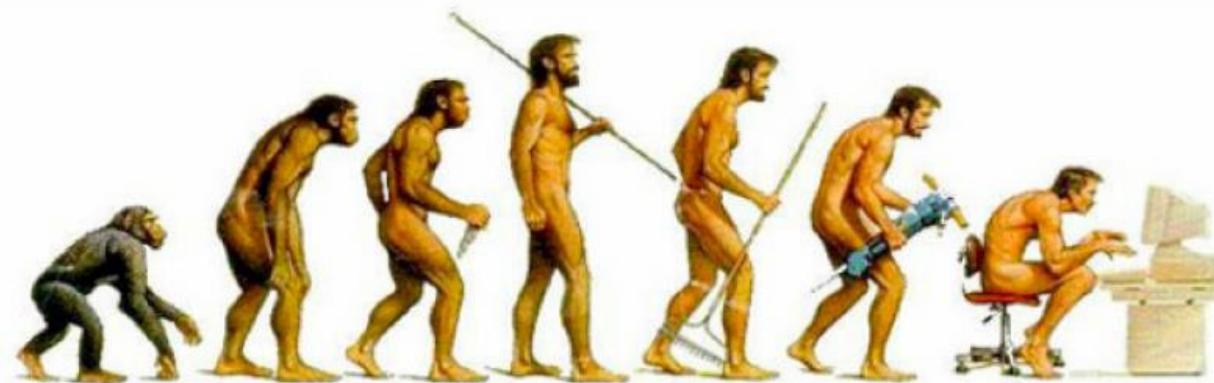
These are technicalities, very important ones, but they are just tools that a person who cares about what he does can use or not in his daily job.

It's not in conflict with the Agile movement. Software Craftsmanship is focused a lot more on technical practices and writing quality software, but the goal is always the same: deliver values for the customer. The only difference is

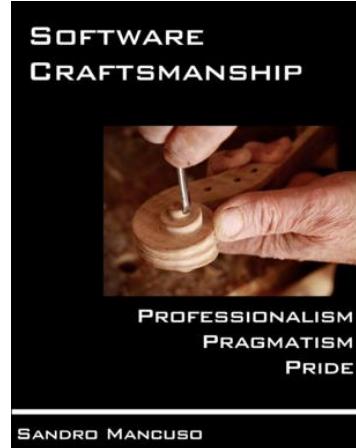
Ligne de production agile (outillage)

Les outils évoluent ...

Les outils évoluent tout comme nous



Etre agile : raccourcir la boucle de feedback



We don't *do Agile*. Either we *are Agile* or we *are not*.

[Being Agile means] adapting successfully to new circumstances—Tom Gilb

- ✓ On ne fait pas de l'agile, on est agile ...
- ✓ **Être agile, c'est raccourcir la boucle de feedback pour pouvoir réagir au plus vite et s'améliorer.**
- ✓ L'agilité ne règle pas les problèmes, elle les met en lumière.

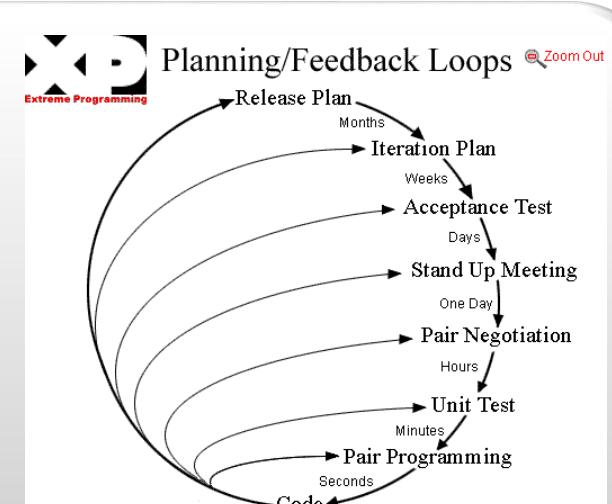
Ligne de production agile ...

Où en est-on ?

- ✓ Adéquation aux besoins
- ✓ Réduction des délais
- ✓ Qualité

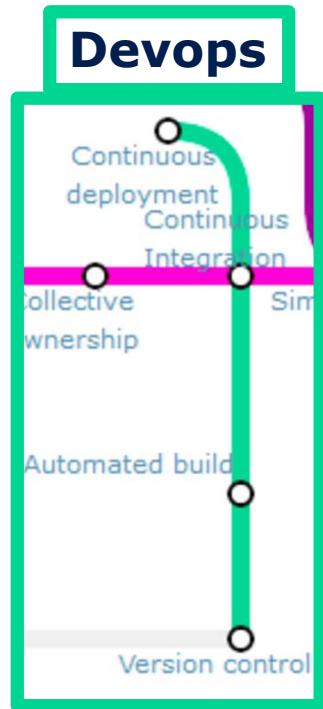
La boucle de Feedback

- ✓ Test en continu
- ✓ Intégration continue
- ✓ Inspection continue
- ✓ Déploiement continu

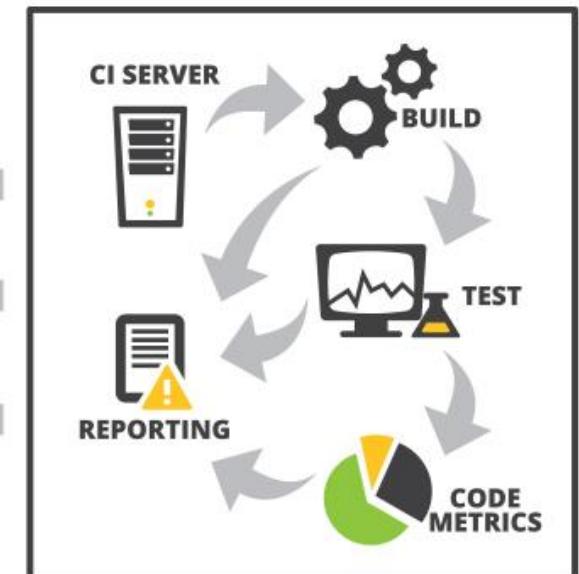
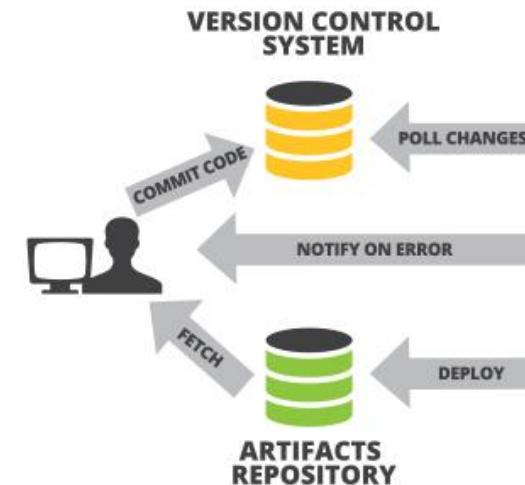


<http://www.extremeprogramming.org/introduction.html>

Processus d'intégration continue



femto-st
SCIENCES & TECHNOLOGIES



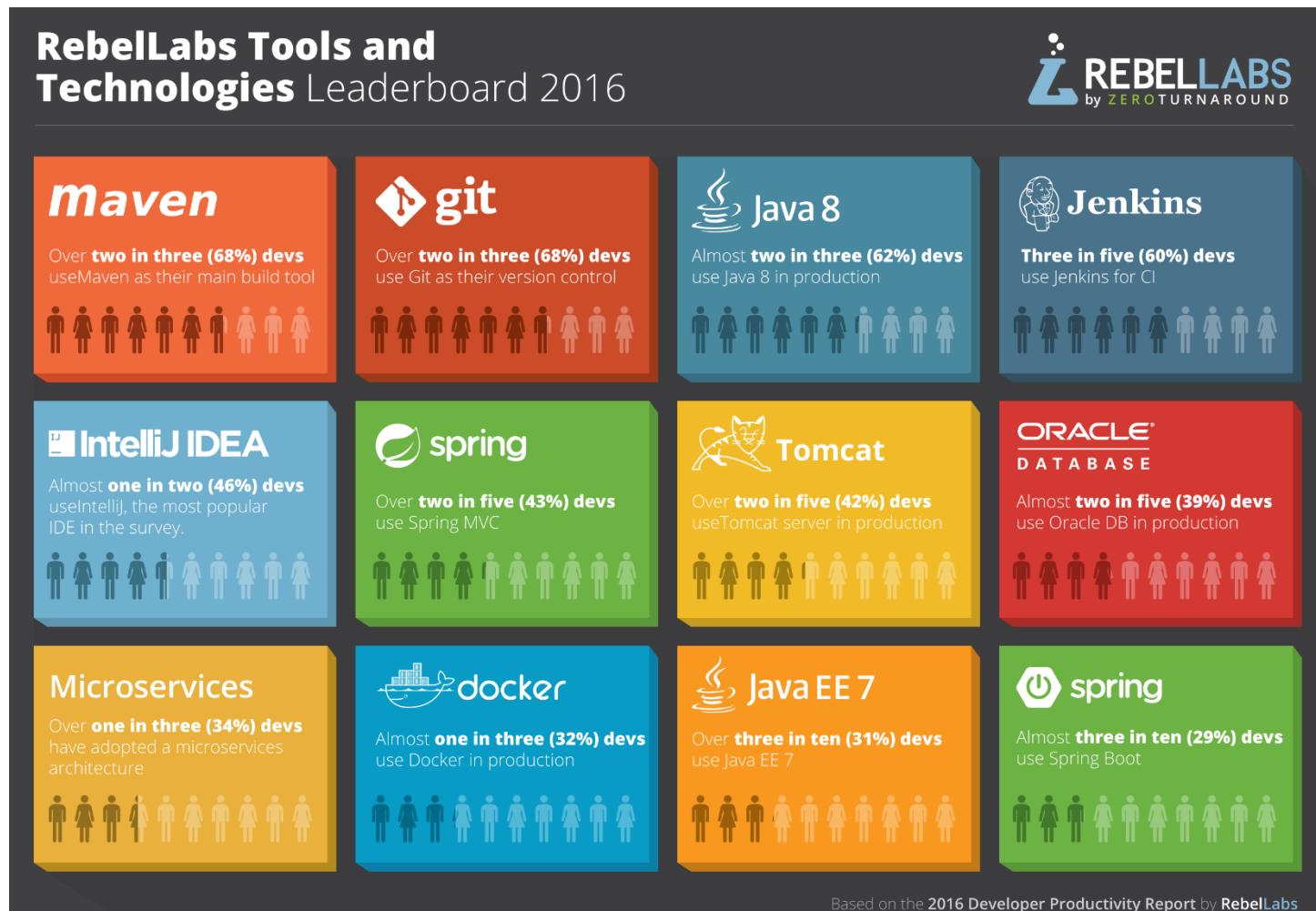
Extraits : http://devlog.cnrs.fr/_media/jdev2013/jdev2013/t6/t6p_jdev13_part2_test-agilite.pdf

http://pages.zereturnaround.com/RebelLabs-AllReportLanders_WhyDevs3CIAGuidetoLovingContinuousIntegration.html

En savoir plus sur l'intégration continue: <https://www.agilealliance.org/glossary/continuous-integration/>

Ligne Devops extraite de : <https://www.agilealliance.org/agile101/subway-map-to-agile-practices>

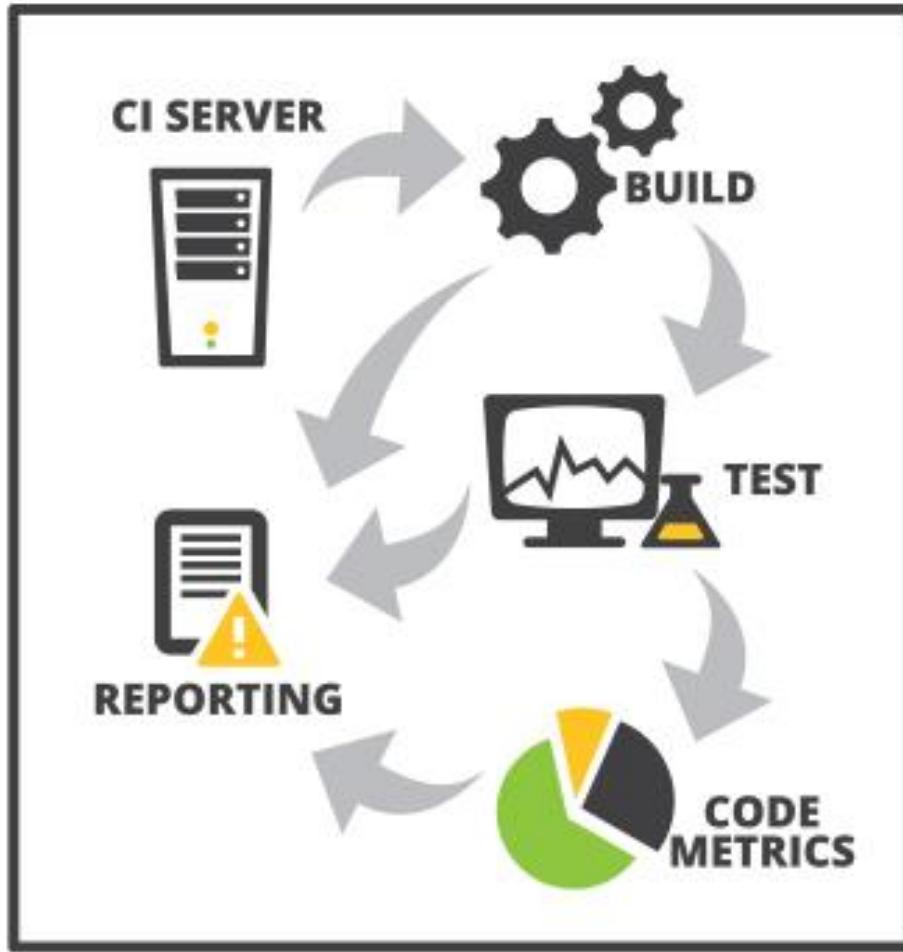
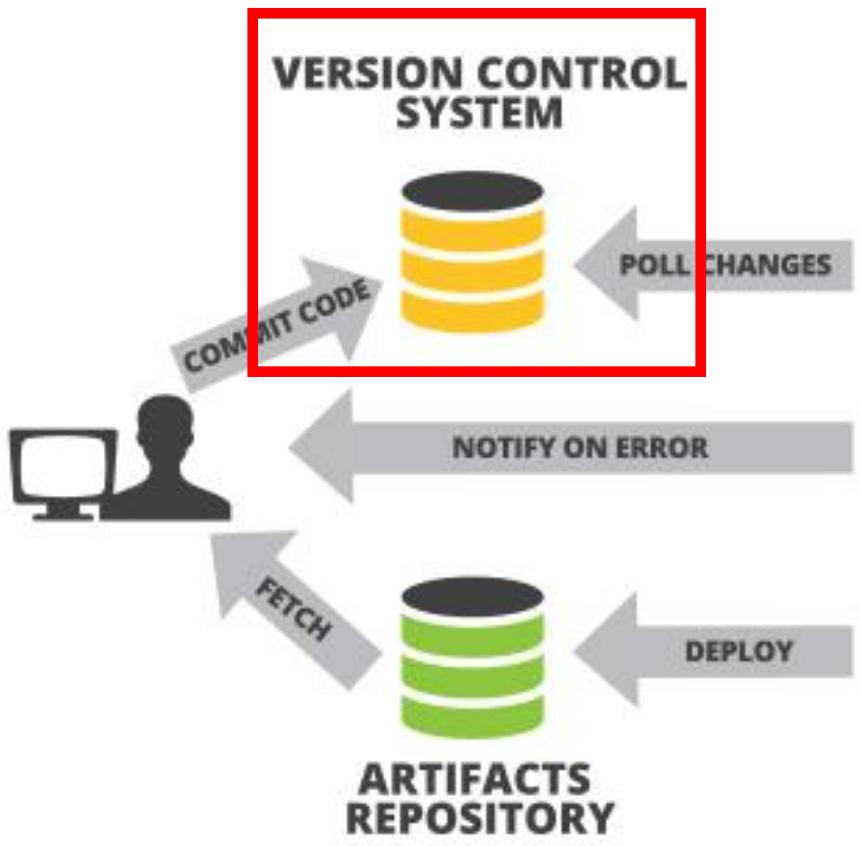
Les données suivantes sont issus de d'un rapport dont les sondages ont réalisés dans un écosystème java



Extraits: <http://pages.zeroturnaround.com/RebelLabs-Developer-Productivity-Report-2016.htm>

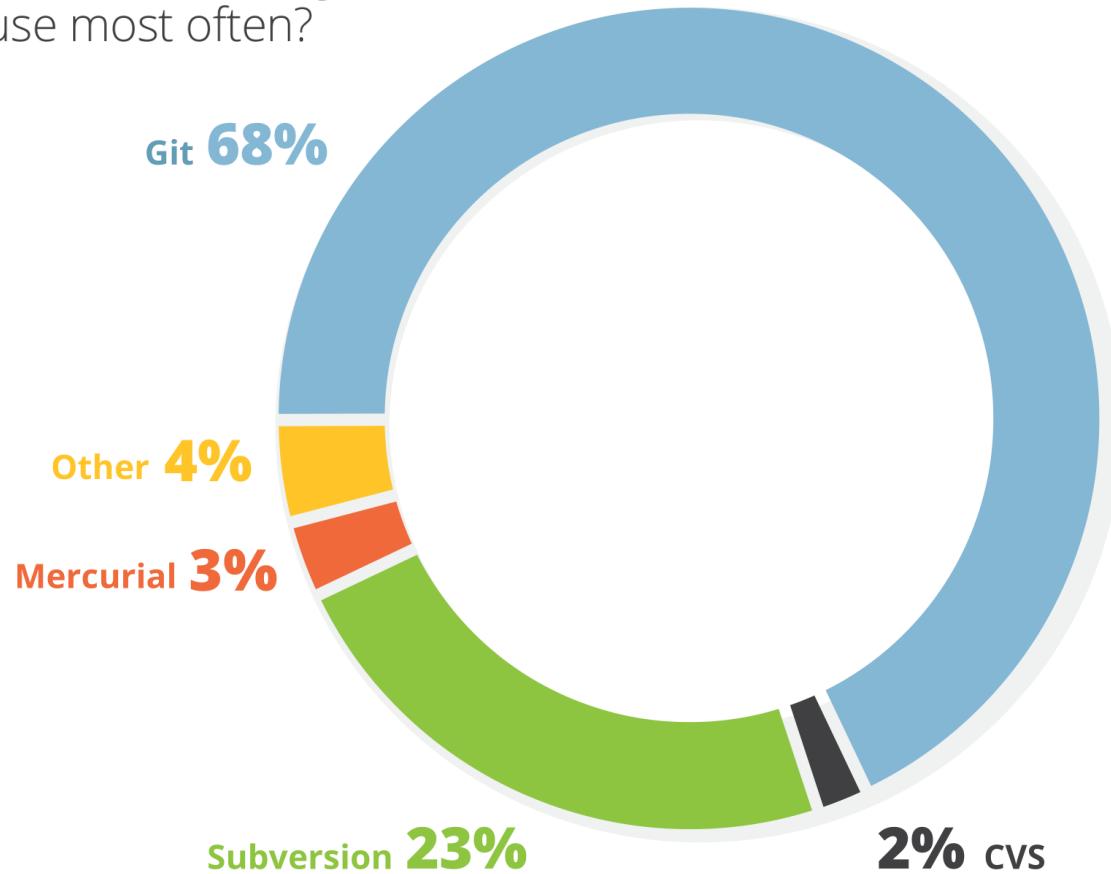
Tous les rapports RebelLabs disponibles sous : <http://zeroturnaround.com/rebellabs/reports/>

Isabelle BLASQUEZ - 2016



L'indispensable Gestionnaires de Version

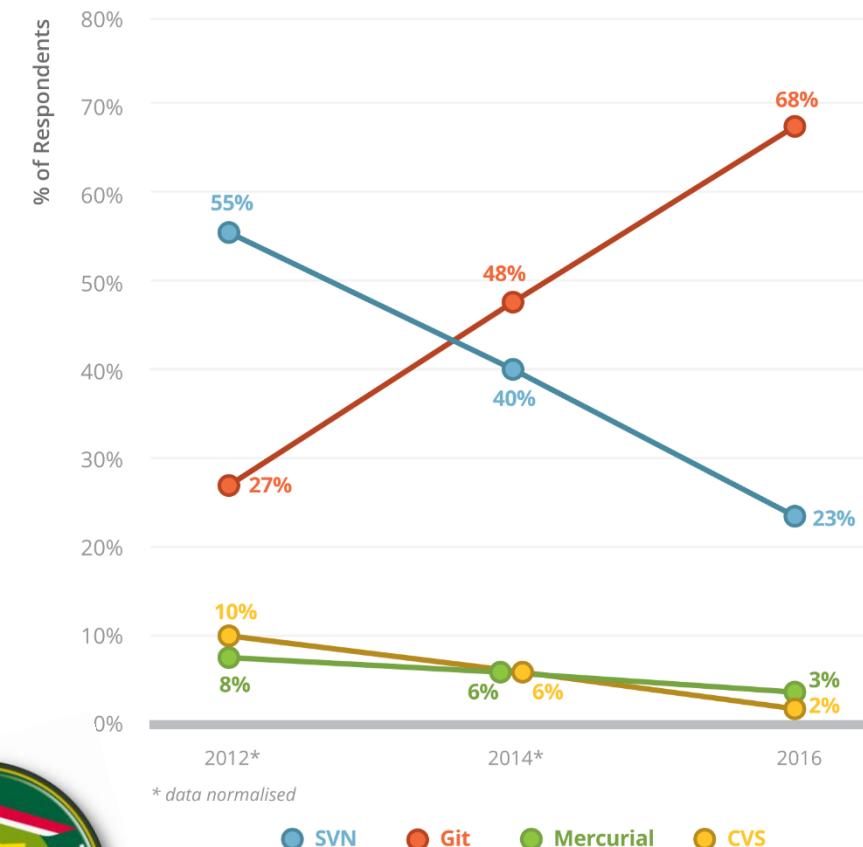
Which VCS do you
use most often?



All rights reserved. 2016 © ZeroTurnaround Inc.

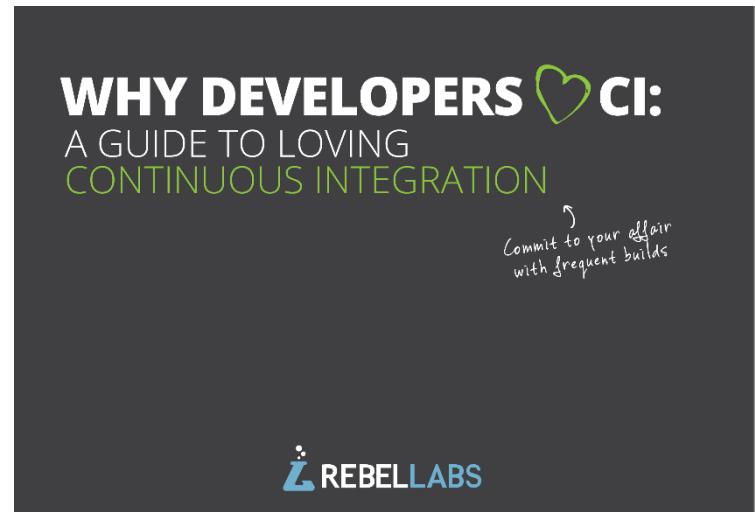
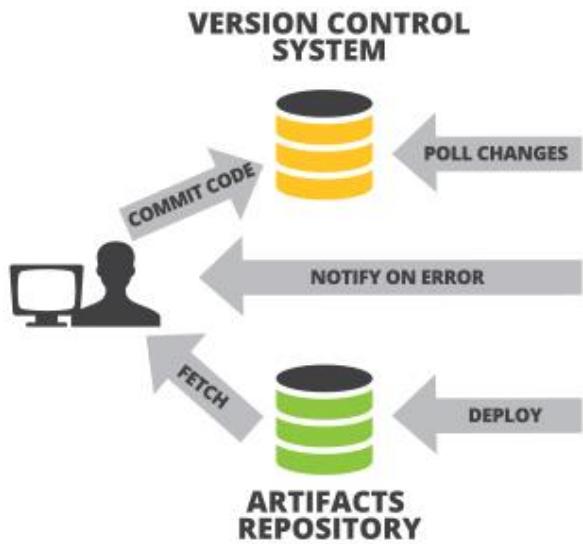
Extrait : <http://pages.zeroturnaround.com/RebelLabs-Developer-Productivity-Report-2016.html>

Figure 3.6 VCS Usage Since 2012



Cyril Lacôte @clacote - 30 sept.
En soutien aux développeurs et à l'artisanat du code source français,
bientôt un label "git de France".

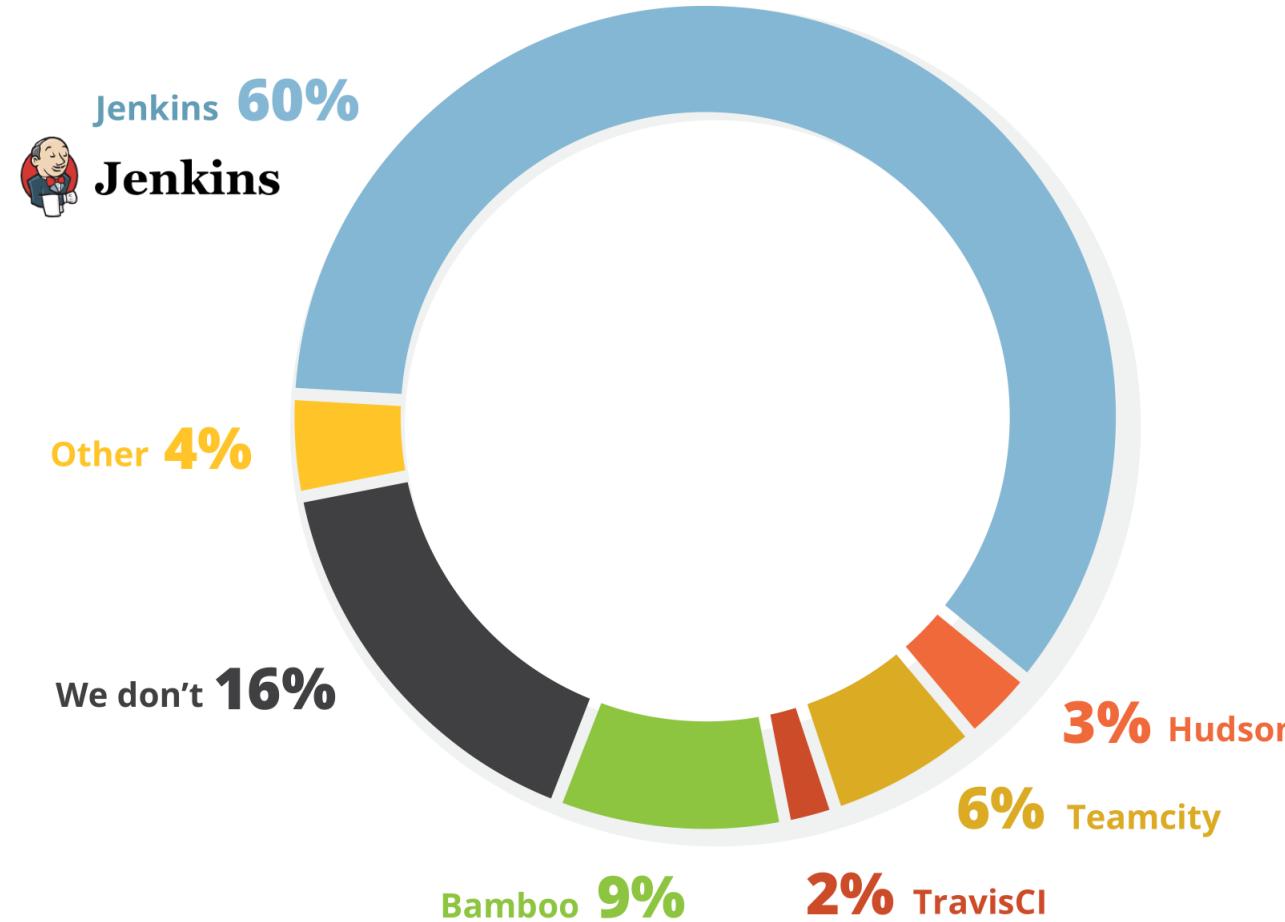
Isabelle BLASQUEZ - 2016



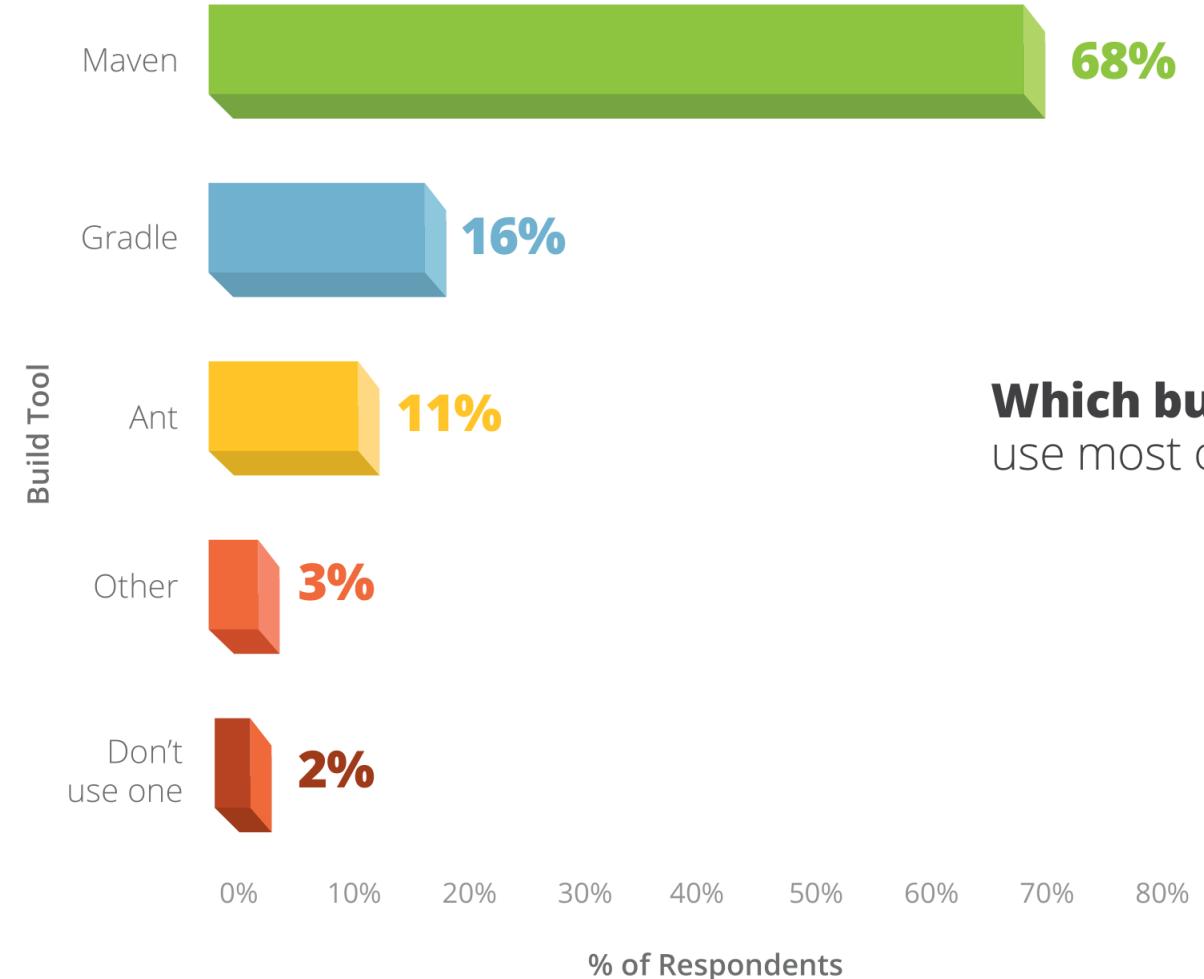
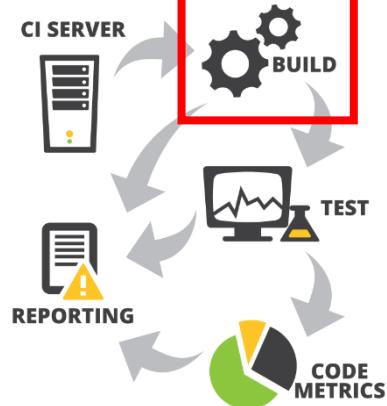
CI SERVERS:	BUILD AUTOMATION TOOLS:	ARTIFACT REPOSITORIES:	TEST FRAMEWORKS:	CODE ANALYSIS:
Jenkins	Maven	Nexus	JUnit	sonar
Hudson	gradle	artifactory	TestNG	
Bamboo	MAKE	archiva	Se	
TeamCity	RAKE			
cruisecontrol				
Travis				

Intégration Continue

Which Continuous Integration Server do you use?



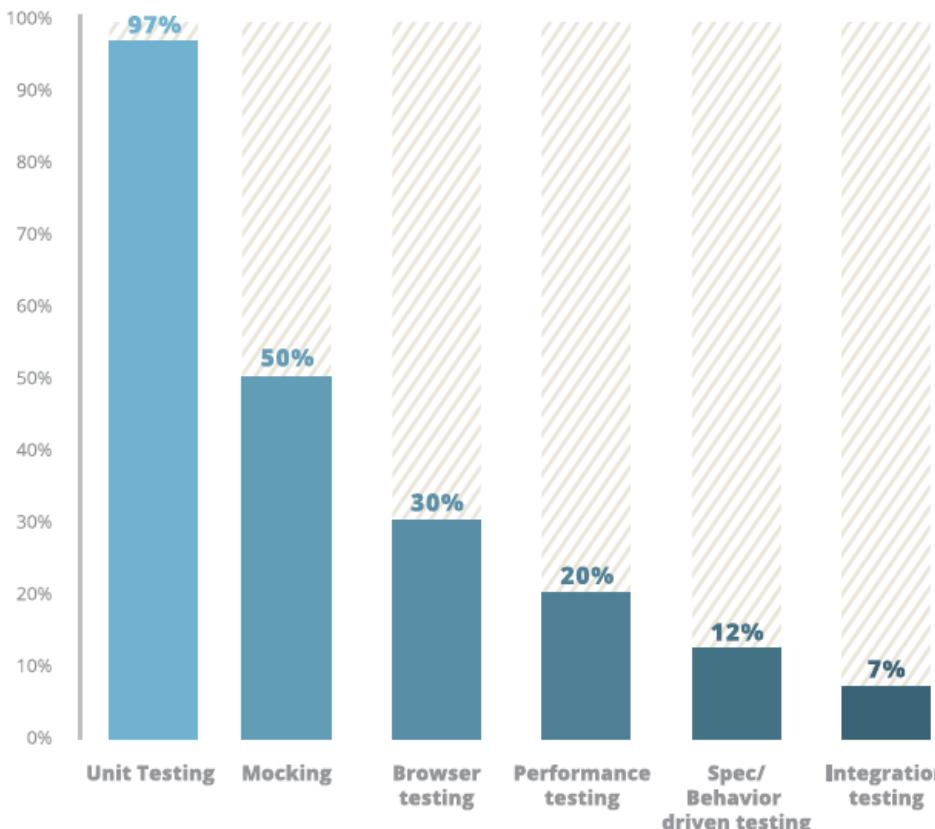
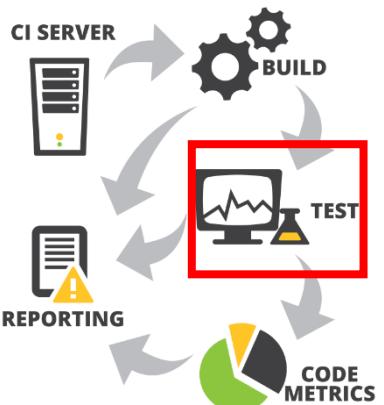
Build



Which build tool do you
use most often?

Test

Types of tests being performed

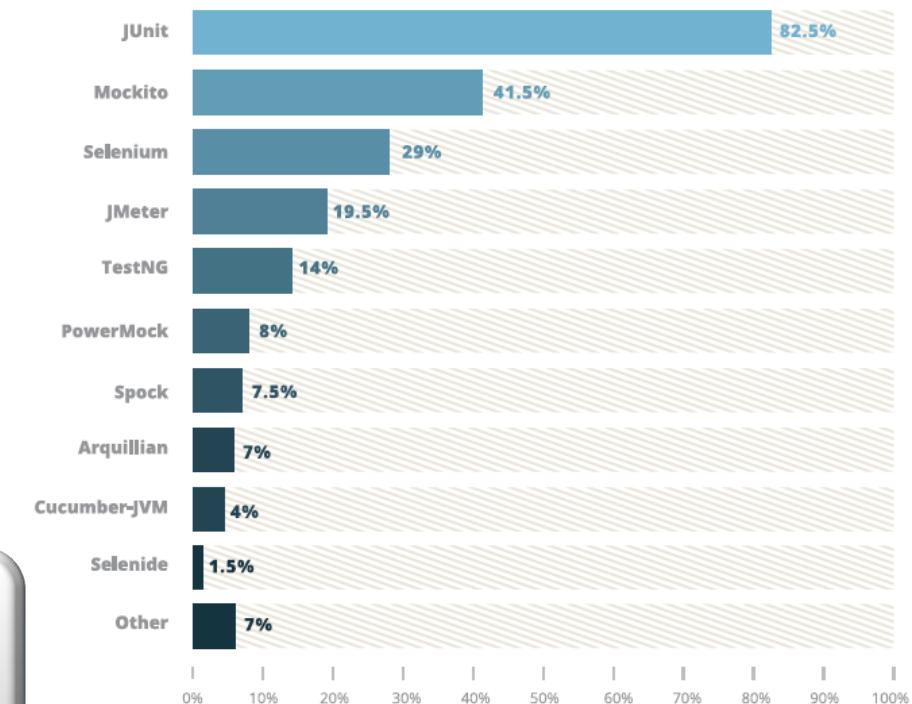


Permet de lancer automatiquement les tests quand un changement est détecté dans le code source
(plugin pour IDE Eclipse ou IntelliJ)

<http://infinitest.github.io/>

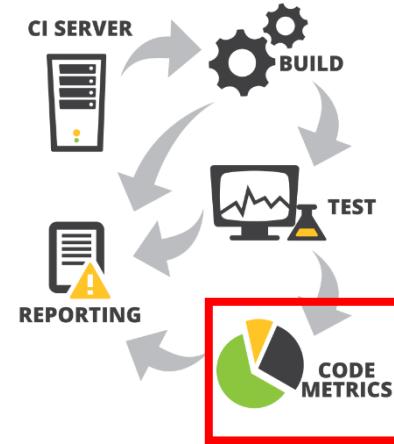
As software producers (and users) it's relieving to see that the vast majority of developers - nearly 97% - do commit to unit testing their apps (i.e. with JUnit & TestNG), and half them apply some kind of mocking framework as well (i.e. Mockito / PowerMock). About 1 in every 3 developers also automate their browser testing (Selenium / Selenide) and 1/5th practise performance and load testing. Integration testing and spec-based or behavior driven testing are not incredibly common still.

Testing technologies in use*

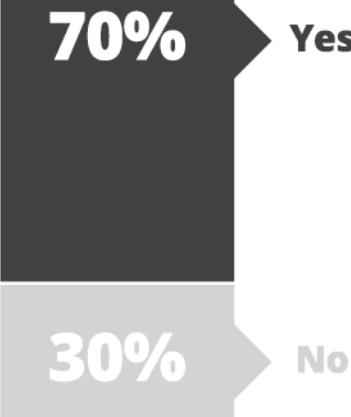


* Multiple selections were possible

Inspection continue

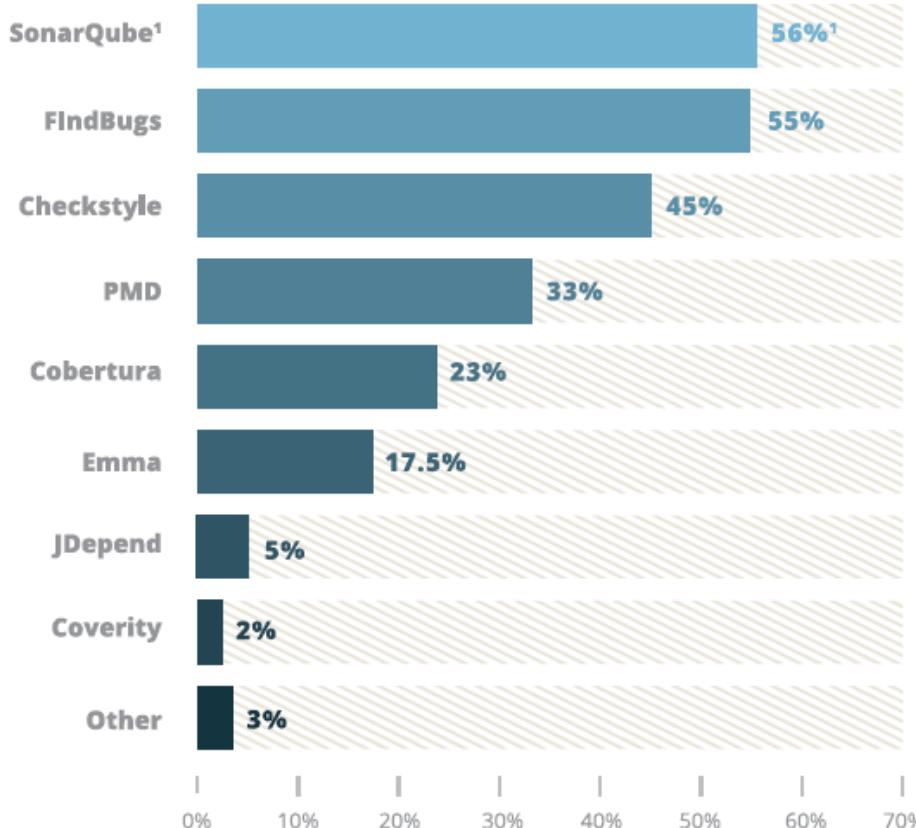


Do you use one or more code analysis tools?



REBELLABS

Code analysis tools in use *



* Multiple selections were possible and the results were normalized to exclude non-users

¹ SonarQube is an integration platform for hosting analysis tools

La mission de Sonar

Permettre de déclarer ouverte la
chasse aux
7 péchés capitaux

Les 7 péchés capitaux

Appliqués au code source

- Duplications
- Mauvaise distribution de la complexité
- Mauvais Design
- Pas de tests unitaires
- Pas de respect des standards
- Bugs potentiels
- Pas ou trop de commentaires

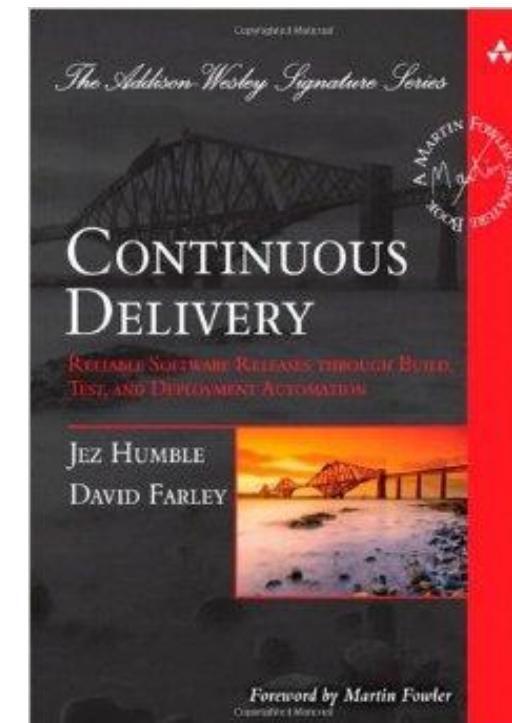
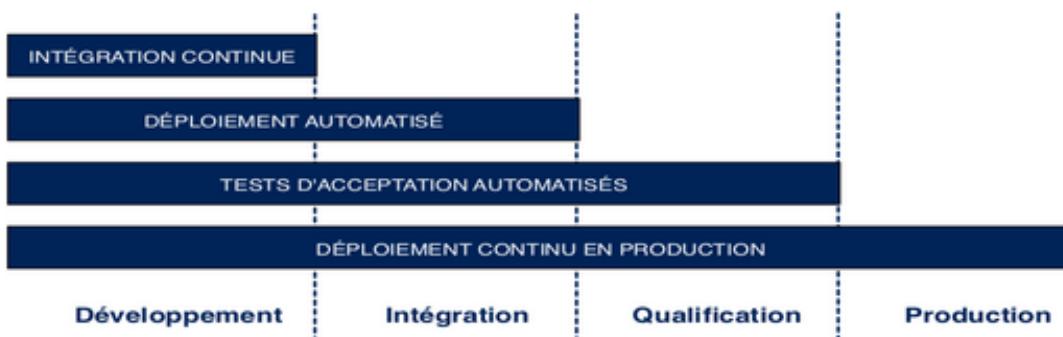
Déploiement continu

DevOps, de l'intégration continue au déploiement continu

Posté le 01/09/2014 - par *Farhdine Boutzakhti, Romain Felden*

Les premiers pas vers une industrialisation des développements consistent généralement en la mise en place d'une intégration continue.

Alors que celle-ci est souvent vue comme un aboutissement, elle n'est qu'une première étape pour parvenir à des réalisations efficaces et maîtrisées.



Article paru dans le magazine **ICT Journal** du mois de Juillet-Août 2014.

Extrait : <http://blog.octo.com/devops-de-lintegration-continue-au-deploiement-continu/>

<http://www.thoughtworks.com/continuous-delivery>

En savoir plus « Définition des pratiques d'intégration continue, de livraison continue et de déploiement continu » de Martin Fowler :

Article original : <http://martinfowler.com/bliki/ContinuousDelivery.html>

Traduction française : http://www.outilfrance.com/index.php?pc=pages/docs/pratique-05/171-04.inc&pb=haut_entete_pratique.inc

Voir aussi la définition : <https://www.agilealliance.org/glossary/continuous-deployment/>

Periodic table of devops tools

Periodic Table of DevOps Tools (v2)																		
Open Source		SCM		Database Mgmt		Build		Testing		Containerization		Collaboration		Security		AWS		
1	Fm	Gh	Github	Os	Sc	Fr	Ci	Fm	Rmgmt	Puppet	An	Sl	Dk	Az	Amazon Web Services			
3	Os	Gt	Git	4	En	Dm	Dbmaestro	Fm	Config / Provisioning	Puppet	Ansible	Salt	Docker	Azure	10	Pd		
11	Fm	Bb	Bitbucket	12	Os	Lb	Liquibase	Pd	Release Mgmt	Vagrant	Terraform	rkt	Tf	Rk	18	En		
19	Os	Gl	GitLab	20	En	Rg	Redgate	Os	Cloud / IaaS / PaaS	Bl	BladeLogic	Vagrant	Terraform	Rk	Gc	Google Cloud Platform		
21	Os	Mv	Maven	22	Os	Gr	Gradle	23	Os	ANT	Jn	Bamboo	Travis CI	31	Pd	32	Os	
24	Os	Fn	FitNesse	25	Fm	Se	Selenium	26	Os	Gating	Dh	Jenkins	Deployment Manager	Sf	Os	33	Os	
37	Os	Sv	Subversion	38	En	Dt	Datical	40	Os	Grunt	Gp	Br	SmartFrog	Cn	Os	34	Os	
39	Os	Gt	Gulp	41	Os	Gp	Broccoli	42	Fm	Cu	Cucumber	Cj	Consul	Bc	Os	35	Os	
43	Os	Sv	Subversion	44	Fm	Npm	npm	45	Os	Qu	Quint	Npm	Bcfg2	Mo	Os	36	En	
46	Fm	Gt	Gulp	47	Pd	Cs	Codeship	48	Fm	Vs	Visual Studio	Cr	Capistrano	Rackspace	Rs	53	Fr	
49	Fm	Hg	Mercurial	50	En	Dp	Delphix	51	Os	Br	Cucumber	Cj	Juju	Rd	Op	54	Os	
52	Os	Hg	Mercurial	53	Fm	Sb	sbt	54	Os	Mk	Make	Ck	CircleCI	Cf	OpenStack	55	Fm	
55	Os	Hg	Mercurial	56	En	Dp	Delphix	57	Fm	Jt	JUnit	Jm	CircleCI	Ds	Swarm	56	En	
58	Os	Hg	Mercurial	59	Os	Sb	sbt	60	Fm	Jm	JMeter	Tn	RapidDeploy	Op	OpenStack	57	Fm	
61	Fm	Hg	Mercurial	62	Fm	Mk	Make	63	Os	Ay	Artifactory	Tc	Octopus Deploy	No	CA Nolio	58	En	
64	Fm	Hg	Mercurial	65	Fm	Sh	Shipable	66	Os	Ry	RapidDeploy	Cc	CodeDeploy	Kb	Kubernetes	59	Os	
67	En	Hg	Mercurial	68	Fm	Tc	TeamCity	69	En	Cy	Octopus Deploy	Oc	Octopus Deploy	Hr	Heroku	60	Fm	
70	En	Hg	Mercurial	71	Os	Sh	Shipable	72	Fm	No	CA Nolio	Ca	CodeDeploy	Ca	Nolio	71	Os	
73	En	Cw	ISPW	74	En	Ild	Idera	75	Os	Msb	MSBuild	Rk	Continuum	Co	Xld	88	En	
76	Os	Cw	ISPW	77	Fm	Pk	Packer	78	Os	Pk	Mocha	Xltv	Jasmine	Ca	EB	89	Os	
79	En	Cw	ISPW	80	Os	Mc	XLTv	81	Os	Nx	Nexus	Continua CI	So	Dp	90	En		
82	Os	Cw	ISPW	83	Fm	Co	Continuum	84	Pd	Xld	XL Deploy	XL Deploy	XD	UrbanCode Deploy	Nm	OpenShift	83	Fm
85	En	Cw	ISPW	86	En	EB	ElasticBox	87	Fm	EB	ElasticBox	ElasticBox	Dp	UrbanCode Deploy	Os	OpenShift	84	Fm
91	En	Xlr	XL Release	92	En	Ur	UrbanCode Release	93	En	Bm	BMC Release Process	94	En	Hp	HP Cedar	95	En	
95	En	Xlr	XL Release	96	En	Pl	Plutora Release	97	En	Sr	Serena Release	98	Pd	Tfs	Team Foundation	99	Fm	
100	Pd	Xlr	XL Release	101	Fm	Jr	Jira	102	Fm	Sl	Slack	103	Fm	Fd	Flowdock	104	Pd	
105	En	Xlr	XL Release	106	Os	Ki	Kibana	107	Fm	Nr	New Relic	108	Os	Ni	Nagios	109	Os	
110	En	Ki	Kibana	111	Os	Zb	Zabbix	112	En	Dd	Datadog	113	En	El	Elasticsearch	114	Fm	
115	Fm	Ki	Kibana	116	Os	St	StackState	117	Os	Sp	Splunk	118	Os	Le	Logentries	119	Os	
119	Os	Ki	Kibana	120	En	Gr	Graylog	121	Os	Sl	Sumo Logic	122	Os	Sn	Snort	123	En	
124	En	Ki	Kibana	125	En	Tr	Tripwire	126	En	Ff	Fortify	127	En	Pv	Pivotal Tracker	128	En	

Carte interactive disponible sur : <https://xebialabs.com/periodic-table-of-devops-tools/>

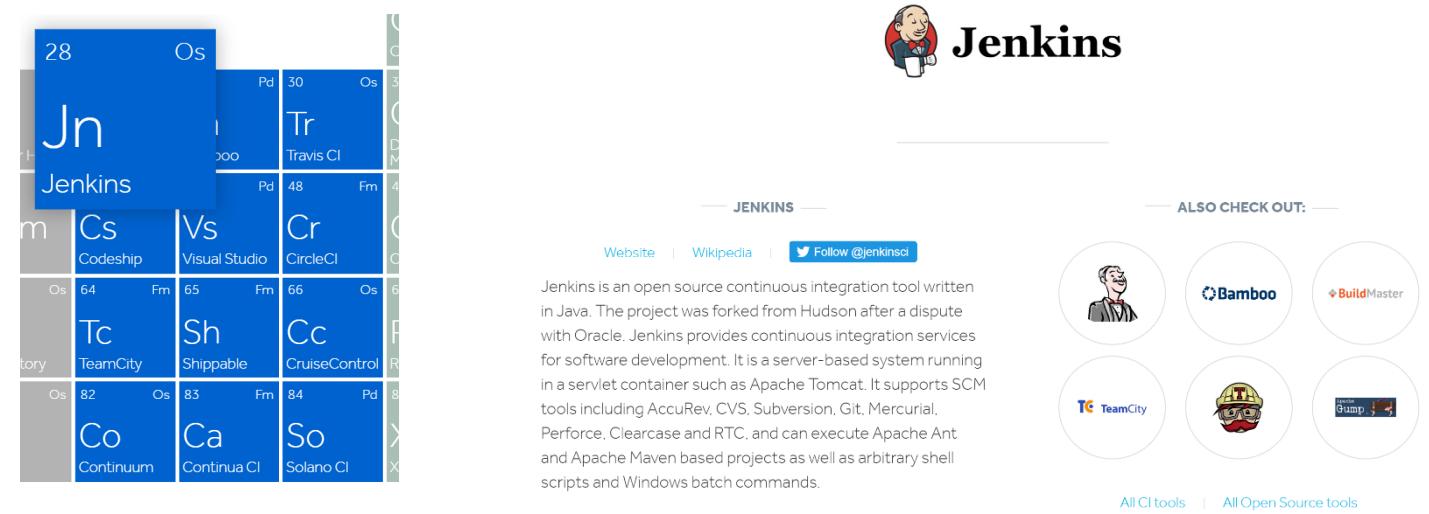


A compléter par :

<https://xebialabs.com/glossary/>

Isabelle BLASQUEZ - 2016

Zoom sur Jenkins (par exemple)



Jenkins

— JENKINS —

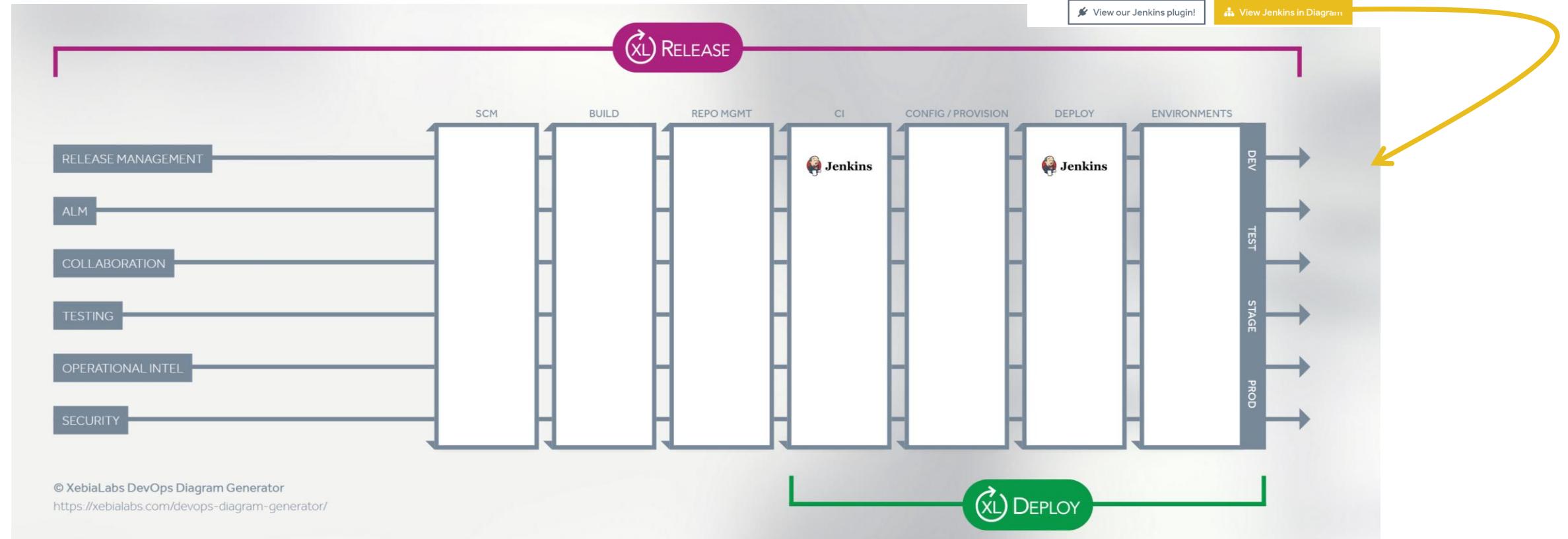
[Website](#) | [Wikipedia](#) | [Follow @jenkinsci](#)

Jenkins is an open source continuous integration tool written in Java. The project was forked from Hudson after a dispute with Oracle. Jenkins provides continuous integration services for software development. It is a server-based system running in a servlet container such as Apache Tomcat. It supports SCM tools including AccuRev, CVS, Subversion, Git, Mercurial, Perforce, Clearcase and RTC, and can execute Apache Ant and Apache Maven based projects as well as arbitrary shell scripts and Windows batch commands.

ALSO CHECK OUT:



[All CI tools](#) | [All Open Source tools](#)



Annexes

Refactoring de code Legacy : des vidéos



Kata de refactoring : Guilded Rose

En Java : Du Legacy au Cloud par David Gageot

Du legacy au Cloud - partie 1/3: <https://www.parleys.com/play/5148922a0364bc17fc56c85a/about>

Du legacy au Cloud - partie 2/3: <https://www.parleys.com/play/5148922a0364bc17fc56c85c/about>

Du legacy au Cloud - partie 3/3: <https://www.parleys.com/play/5148922a0364bc17fc56c85e/about>

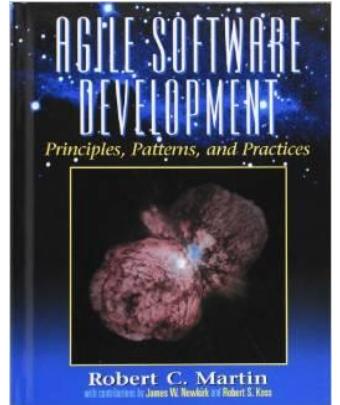
En C# : Refactoring de code legacy avec des trucs en N par Thomas Pierrain

<https://www.youtube.com/watch?v=I3rNyxnD7as>

Kata de refactoring : Trip service

Testing and Refactoring Legacy Code par Sandro Mancuso: https://www.youtube.com/watch?v=_NnEIPO5BU0

SOLID : des références



- ✓ Dans le livre Agile Software Development, Principles, Patterns and Practices, Robert C. Martin a condensé, en 2002, 5 principes fondamentaux de conception, répondant à la problématique d'évolutivité du système , sous l'acronyme **SOLID**

(Extrait : <http://blog.xebia.fr/2011/07/18/les-principes-solid/>)

- ✓ Les articles originaux **SOLID** sur le blog de Robert C. Martin (Uncle Bob) :

<http://butunclebob.com/Articles.UncleBob.PrinciplesOfOOD>

http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf

<http://www.objectmentor.com/resources/articles/srp.pdf>

<http://www.objectmentor.com/resources/articles/ocp.pdf>

<http://www.objectmentor.com/resources/articles/lsp.pdf>

<http://www.objectmentor.com/resources/articles/isp.pdf>

<http://www.objectmentor.com/resources/articles/dip.pdf>

- ✓ Des articles plus récents sur **SOLID** par Uncle Bob :

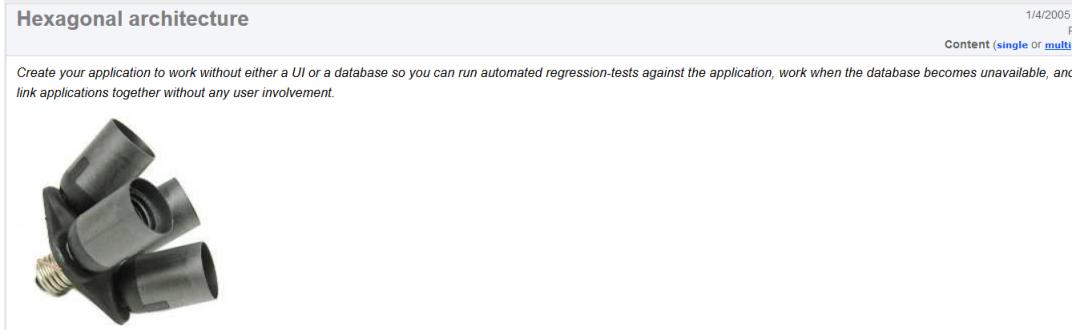
<http://blog.8thlight.com/uncle-bob/2014/05/08/SingleResponsibilityPrinciple.html>

<http://blog.8thlight.com/uncle-bob/2014/05/12/TheOpenClosedPrinciple.html>

- ✓ De nombreux articles sont disponibles sur le web ...

- ✓ Videos : <https://www.youtube.com/watch?v=TAVn7s-k09o> (SOLID Javascript)

En savoir plus sur l'architecture hexagonale et les micro-services ...



The Pattern: Ports and Adapters ("Object Structural")

Alternative name: "Ports & Adapters"

Alternative name: "Hexagonal Architecture"

Intent

Allow an application to equally be driven by users, programs, automated test or batch scripts, and to be developed and tested in isolation from its eventual run-time devices and databases.

As events arrive from the outside world at a port, a technology-specific adapter converts it into a usable procedure call or message and passes it to the application. The application is blissfully ignorant of the nature of the input device. When the application has something to send out, it sends it out through a port to an adapter, which creates the appropriate signals needed by the receiving technology (human or automated). The application has a semantically sound interaction with the adapters on all sides of it, without actually knowing the nature of the things on the other side of the adapters.

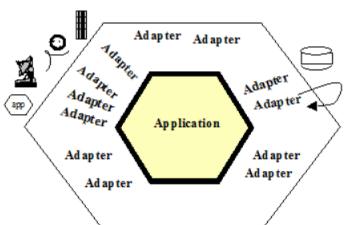


Figure 1

La suite de l'article sur <http://alistair.cockburn.us/Hexagonal+architecture>



Microservices

The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.

La suite de l'article sur <http://martinfowler.com/articles/microservices.html> avec de nombreux liens et références