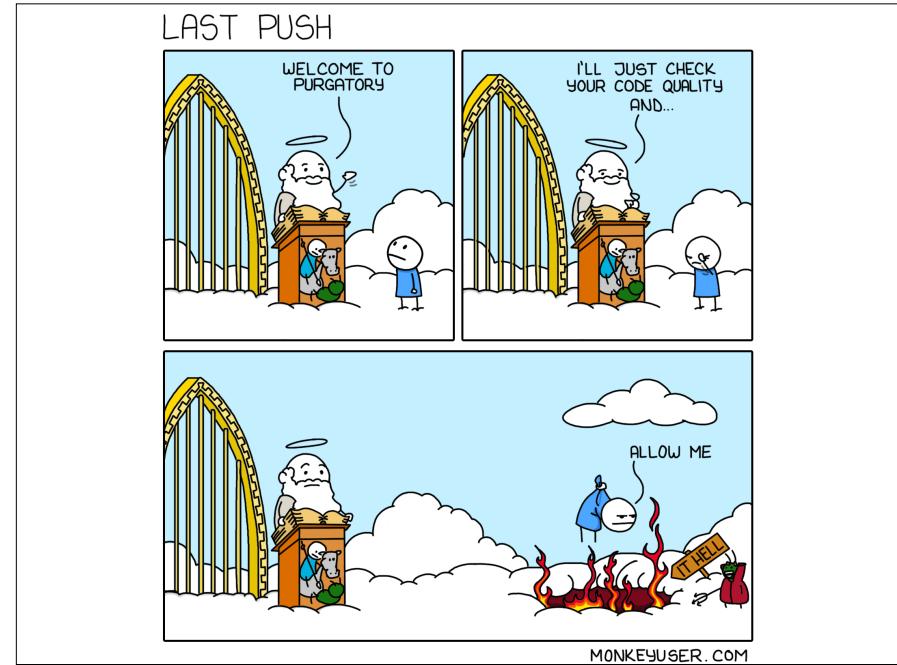
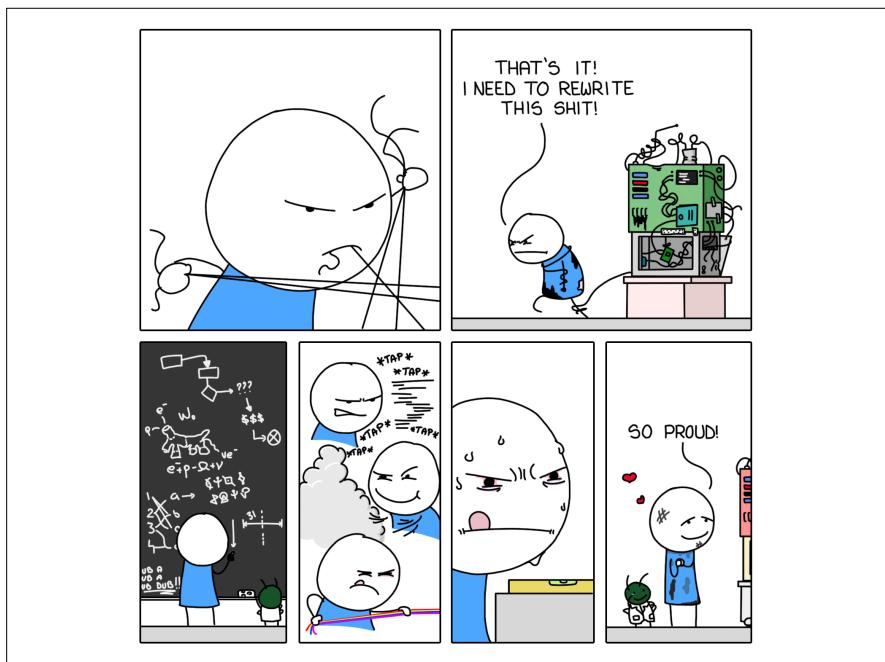




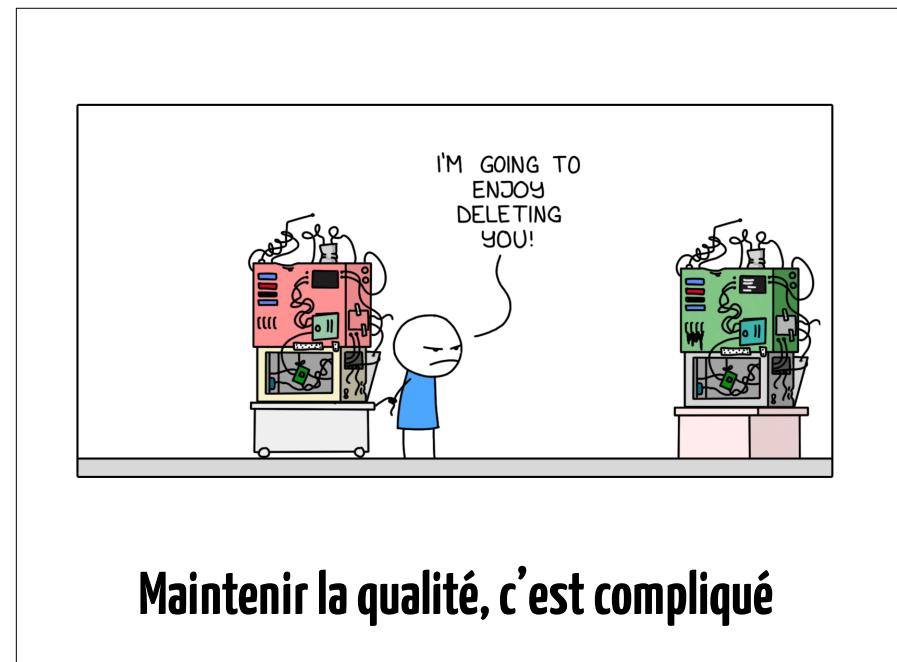
1



2



3



Maintenir la qualité, c'est compliqué

4

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

5

“Ça va sans le dire !”

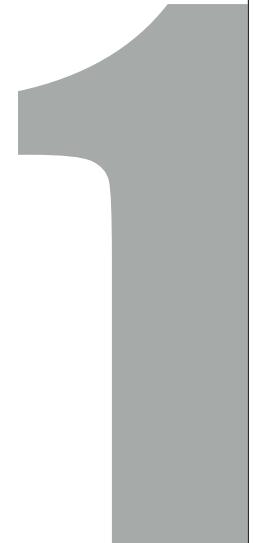
**Mais c'est toujours
mieux en le disant.**

6

- 1 Anti-patrons de Code
- 2 Anti-patrons de Conception
- 3 Anti-patrons d'Architecture
- 4 Anti-patrons d'Équipe

7

**Anti-Patrons
(code objet)**



8

Le code “**objet-pas-objet**”

- Sur-utilisation du procédural
- Explosion de la Loi de Demeter (“principe de connaissance minimale”)
- Le faux objet

“C'est de l'objet, c'est écrit en Java”

9-1

I’héritage, ce truc mystérieux

- Sur-utilisation de l’héritage
- Sur-utilisation des classes utilitaires
- Absence d’utilisation du polymorphisme

“C'est de l'objet, y'a de l'héritage”

10-1

Le code “**objet-pas-objet**”

- Sur-utilisation du procédural
 - “Static everywhere”, nombres magiques dans le code
- Explosion de la Loi de Demeter (“principe de connaissance minimale”)
 - Object o = a().b().c().d().e().f().g().h().i().j().k().l().m().n().o().p().q().r();
- Le faux objet
 - Rien n'est statique, mais il existe un seul objet dans le système ...
 - L'objet est une décomposition fonctionnelle (un processus, ComputeXXX)

“C'est de l'objet, c'est écrit en Java”

9-2

I’héritage, ce truc mystérieux

- Sur-utilisation de l’héritage
 - Classes filles vides et / ou inutiles
 - Héritage utilitaire, mais sans relation “is-a” (classe hérite de Helper)
- Sur-utilisation des classes utilitaires
 - Tout est dans des helpers, pas de responsabilité dans les classes
- Absence d’utilisation du polymorphisme
 - Des ifs / switch / instance of de partout dans le code

“C'est de l'objet, y'a de l'héritage”

10-2

L'abstraction de la mort

- Aussi connu sous le nom de sur-généralisation
- Ou aussi d'optimisation prématuée dans la conception
- Quel en est l'impact ?

“C'est de l'objet, y'a des abstractions”

11-1

**Anti-Patrons
(conception)**



12

L'abstraction de la mort

- Aussi connu sous le nom de sur-généralisation
 - Classes intermédiaires vides et / ou inutiles
 - Classes abstraites jamais héritées, ou avec un seul enfant
- Ou aussi d'optimisation prématuée dans la conception
 - C'est pour ça qu'on conçoit de manière itérative
 - Quel en est l'impact ?
 - Infernal à utiliser, et consomme énormément de temps au début

“C'est de l'objet, y'a des abstractions”

11-2

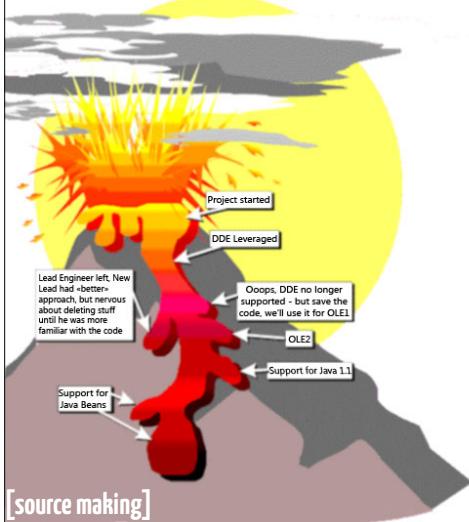
Votre Premier Anti-Patron

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



13

Le fleuve de lave



Mauvaises
décisions
prises

Code “mort”
dans la base
de code

14

Le poltergeist

Classes inutiles
(sans responsabilité)

Cycle de vie
(ultra court)

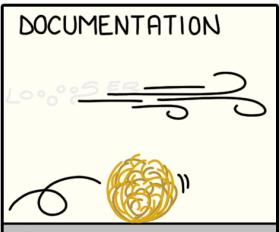
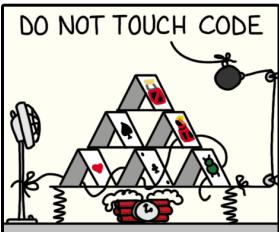
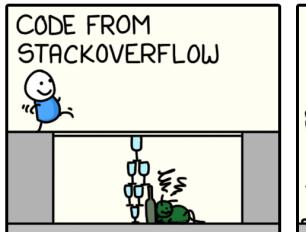


15

Le champ de mine

Bug mineur mais conséquences majeures

Absence de maîtrise des objets conçus



16

Et tous les autres

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

17

Anti-Patrons (architecture)

3

18

“Design by committee”

Mettre tout le monde d'accord

Trop de **compromis**

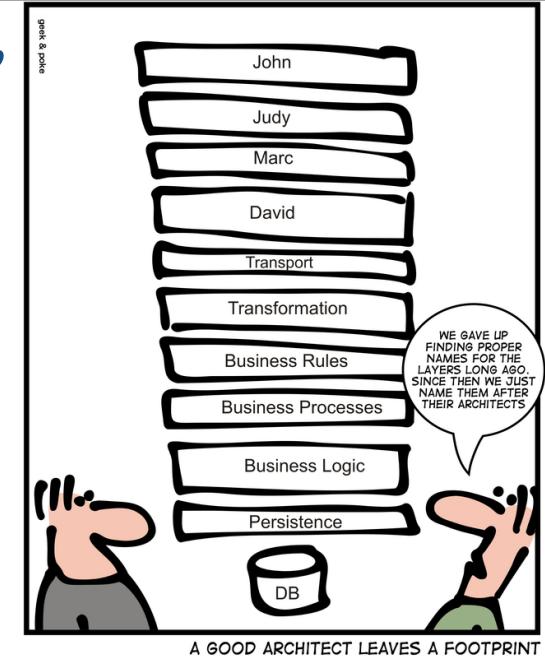
Lenteur / **sur-documentation**



20

“Empilement”

N-tiers
(avec N grand)



19

“The Grand Old Duke of York”

Capacité **Abstraction** ≠ Capacité d'**implémentation**

Un processus de conception égalitaire amène les capacités d'abstraction de l'équipe au PPCM des capacités de chacun !

Il faut savoir faire confiance.



21

“Le produit Couteau-Suisse”

Ajout de **fonctionnalités non cohésives**



Manque de responsabilisation des commerciaux

Aussi appelé
“architecture balle de golf”

22

**Anti-Patrons
(équipe)**



24

“Réinvention de la roue”

TOO BUSY TO IMPROVE?



Faire son propre cache, sa propre BD, ...

23

Paralysie d’analyse

Analyser chaque nouvelle fonctionnalité **pendant des jours/mois/années**.

Ne pas ajouter de valeur au projet pendant cette période. **Sur-modélisation** assurée.

25

Miroir de Fumée

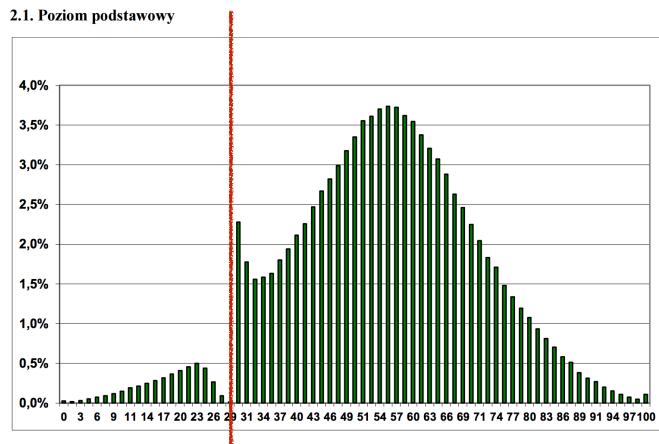
N'implémenter
que ce qui à
été acheté !

Utiliser des
magiciens d'Oz
en démo avec les clients



26

Management par les métriques



<https://sahandsaba.com/nine-anti-patterns-every-programmer-should-be-aware-of-with-examples.html>

28

Bikeshedding

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

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

Loi de la trivialité (Parkinson)

Berkeley Software Distribution (aka BSD)

27

“Signal d’alarme”-driven development



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

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

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

29