« Clean Code » ?

Il s’agit de code.

On parle de **code**, non du code d’une personne, ni d’une personne.

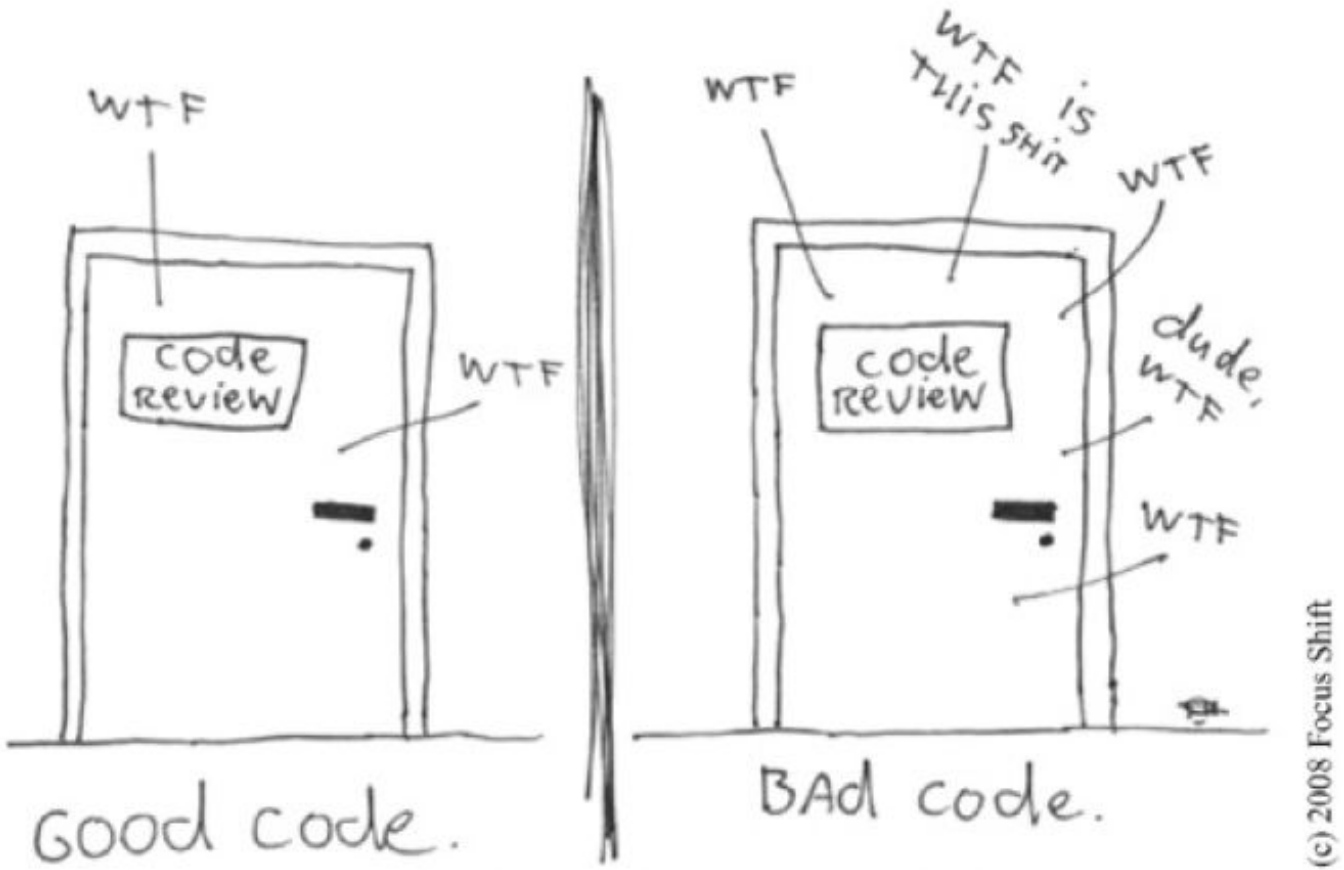
« bon code » vs « bad code » ?

Il faut définir des **critères de qualité** avant de répondre.

Un code productif utilisé et qui fonctionne a de la valeur.

Pas de sur-qualité sans critères de qualité.

Un point de vue…



Reproduced with the kind permission of Thom Holwerda.
http://www.osnews.com/story/19266/WTFs_m

lisibilité > intention > compréhension > maîtrise > sérénité, (tolérance)

L’intention d’un morceau de code doit être **explicite**.

Un code doit pouvoir **se lire** tel une histoire, celle du domaine.

Un code lisible coûte **moins cher**.

Où est passé le *métier*?

Un développeur parle avec un BA dans un langage métier, puis il réalise l'exigence dans son langage de programmation…

Le code devrait fidèlement et durablement **raconter le métier**.

Ecart non-maîtrisé = perte d'information = nouvel investissement.

Evolutivité, maintient, maîtrise > durabilité

La capacité d’un code à être entretenu et à grandir définissent sa **durabilité**.

Un code durable coûte **moins cher**.

Sérénité et économie, par quoi commencer?

Définir (retrouver) quelques principes: une petite **charte**.

Réfléchir avant de coder, **partager** la réflexion, demander le **feedback**.

Penser « communautaire ».

« Clean Code » !

Quelques **principes** de **rédaction** de code et quelques principes de **design** logiciel peuvent nous aider à maîtriser la croissance de notre application.

*“**Clean code can be read, and enhanced by a developer other than its original author**.*

It has unit and acceptance tests. It has meaningful names.

It provides one way rather than many ways for doing one thing.

It has minimal dependencies, which are explicitly defined, and provides a clear and minimal API.

***Code should be literate** since depending on the language, not all necessary information can be expressed clearly in code alone.”*

Lisibilité: principe I

Utiliser des noms significatifs.

Le vocabulaire du domaine.

Des groupes de mots s’il le faut.

Expliciter l’intention.

Eviter les mots au sens trop large.

Lisibilité: principe II

Rédiger de petites méthodes:

5-10 lignes de code au plus

« Keep It Simple »

Lisibilité: principe III

1 méthode, 1 chose

1 méthode, 1 degré d'abstraction

« The Stepdown Rule »

« Keep it simple »

Evolutivité, testabilité: principe IV

SOLID

Single Responsibility Principle

Open-Close Principle
(ouvert à l'extension, fermé à la modification)

Evolutivité, testabilité: principe V

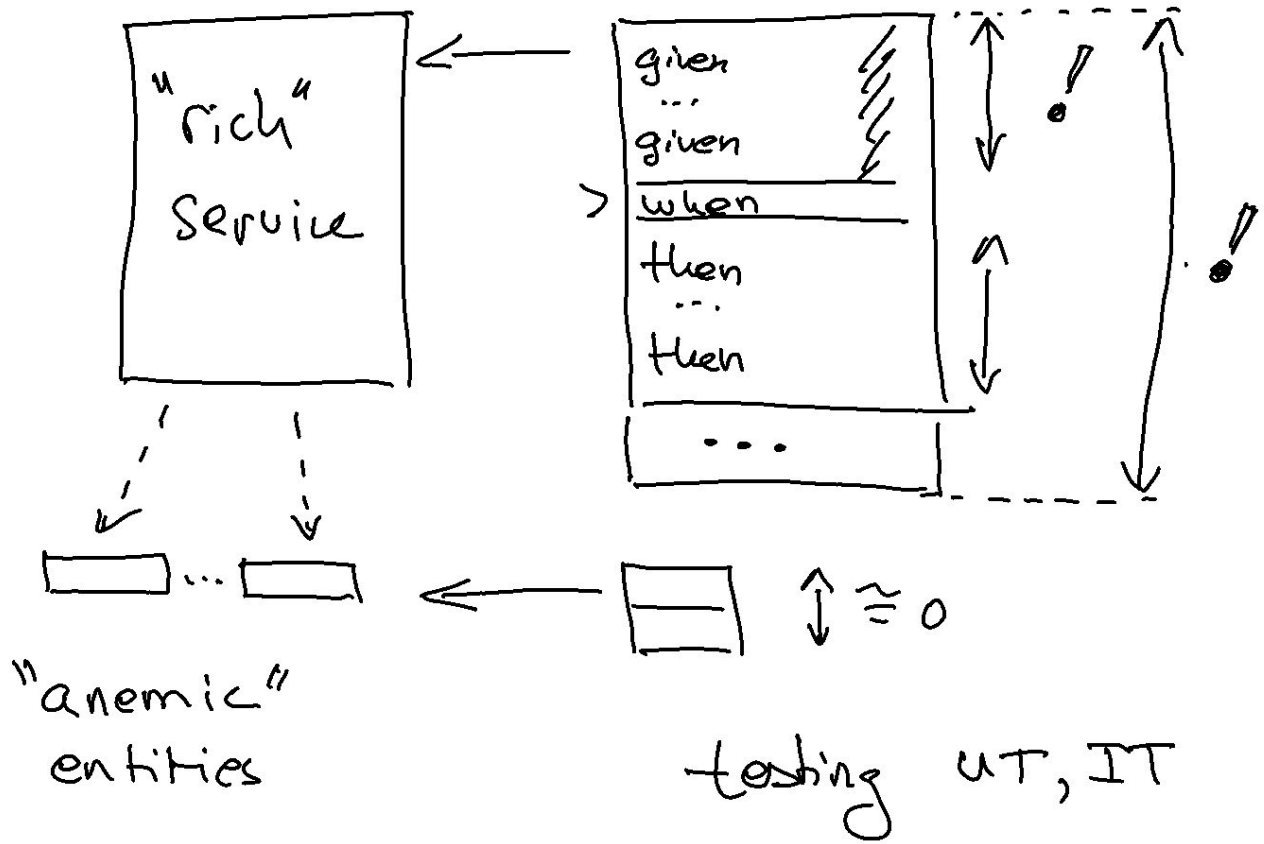
Diviser pour régner, répartir la logique: OOD !

Responsabiliser une population de classes d'objets au comportement spécifique
et moins une petite bourgeoisie de services riches.

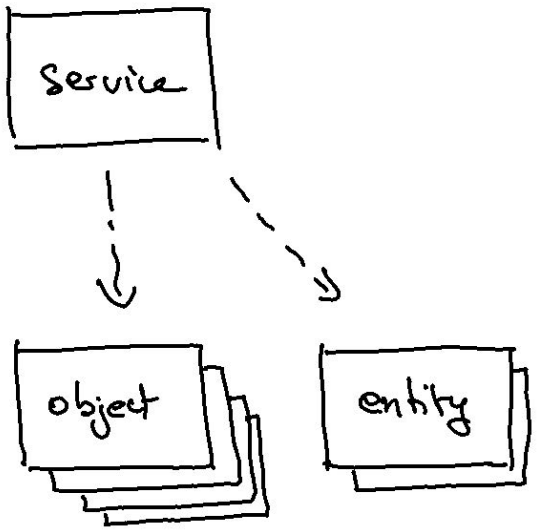
Indicateurs: Loi de Demeter, nombre de dépendances,
taille partie préparatoire des tests, duplication de code

«Le syndrome du service riche»

«Les tests engraissent»



«Une répartition équitable des responsabilités»

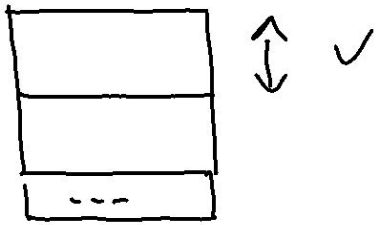


OOO ✓

orchestration ✓

délégation ✓

1 objet : 1 responsabilité



testing : UT, IT

«Les tests comme indicateurs de poids»

« **A good developer** is a developer who can write code that any developer can understand »

Sandro Mancuso

« **Le bon développeur** est bien plus qu'un codeur, il est également apte à construire du code et à en assurer la qualité »

Jean-Pierre Lambert

« **Le bon développeur** sait produire du code qui ne coûte pas cher »