

# Formation Revue de Code

29/03/22


# Special remote

- Deux outils pour cette formation

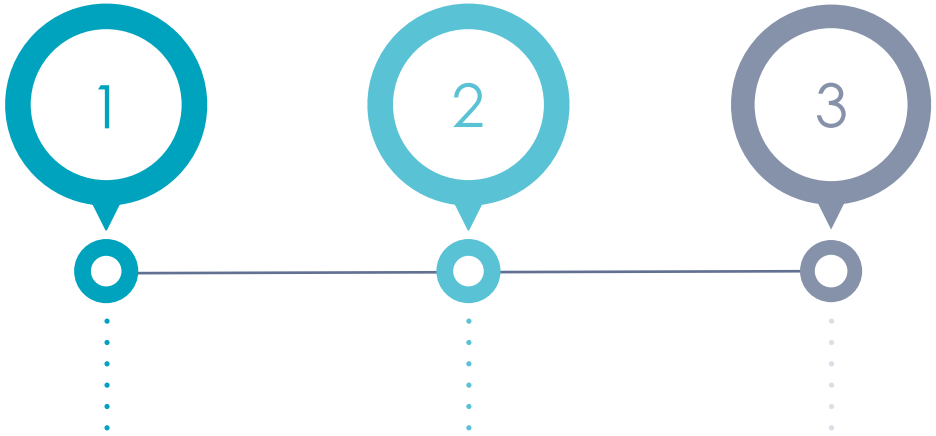
**Visio-conférence**
Pour échanger en plénière
en binômes ou en sous-groupes,
en audio ou avec les webcam

**Un espace de travail partagé**
sous la forme d'un tableau blanc

# Pour commencer

- Horaires ?
  - Jour 1 : 9h30 - 12h - 13h30h - 17h30
- Pauses ? 11h - 15h30

# Agenda



# Icebreaker

# Tour de table

5 minutes

- Qui suis-je ?
- Ai-je déjà pratiqué la revue de code ?
- Mes attentes par rapport à la formation ?



# Qu’est ce que la revue de code ?

# A quoi sert la revue de code ?

# A quoi sert la revue de code ?

- Détecter des défauts plus tôt
- Garantir le respect des standards

**Mais aussi**

- Favoriser la propriété collective du code
- Permettre l'apprentissage / transfert des connaissances
- Améliorer la qualité des échanges
- Diffuser une culture de la qualité
- Réduire la nécessité de corriger des défauts
- Faciliter la correction des défauts
- Réduit la pression sur la QA
- Améliore la satisfaction utilisateur

# Pourquoi ?

- L'erreur est humaine
- Comme pour l'écriture, beaucoup d'erreurs échappent à l'auteur alors qu'elles sautent aux yeux des lecteurs
- On n'écrit pas le code pour la machine qui l'exécute mais pour le développeur qui le lira dans le futur. Le faire relire par un tiers dès aujourd'hui augmente les chances que ce développeur, demain, soit en mesure de le reprendre, de le faire évoluer.

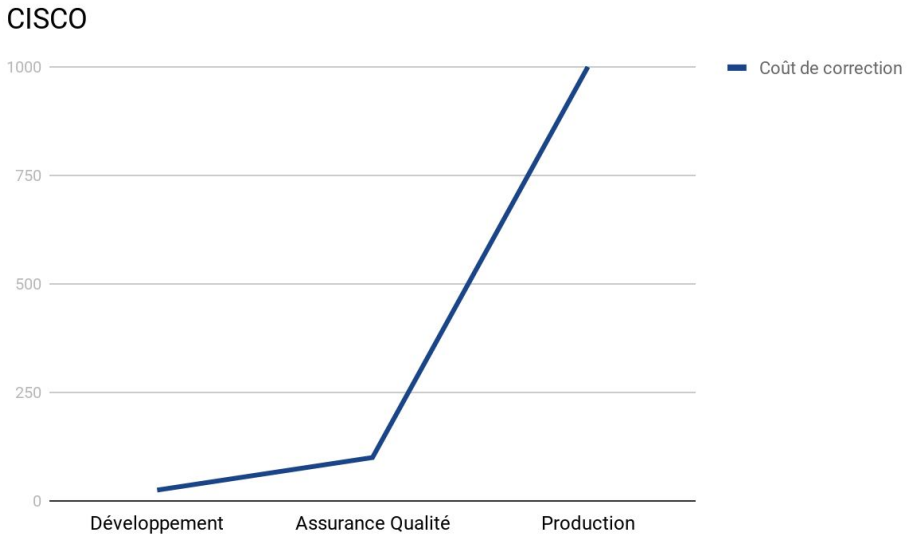
# Effets d'un programme de revues

*"Chaque passe de test trouve 30-35% des défauts, les revues trouvent environ 85% des défauts"*

***– Capers Jones***

**ex. CISCO**

- 50% de réduction des défauts
- coût de correction d'un défaut:
  - En développement: 25
  - En Assurance/Qualité: 100
  - En Production: 1000



# Différences entre les types de revues

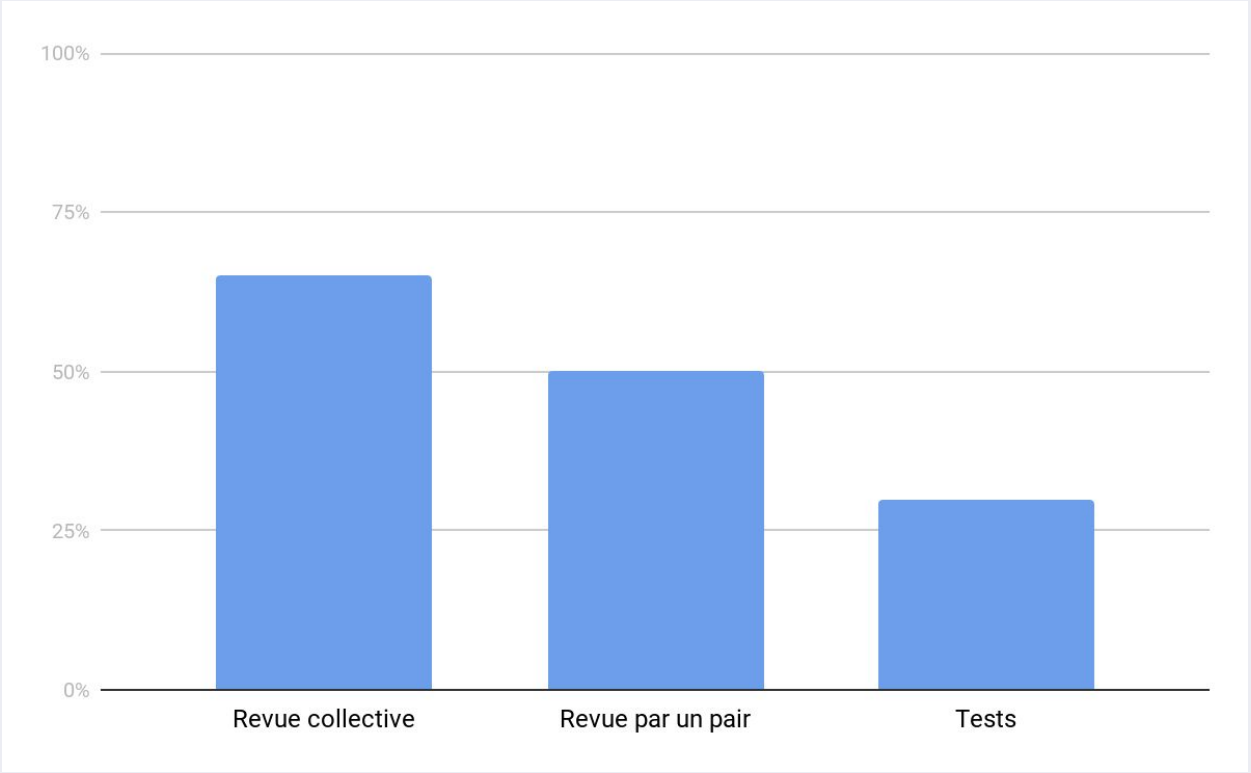
**DÉFAUTS DÉTECTÉS**

**65%**
**EN REVUE COLLECTIVE**

**50%**
**EN REVUE PAR UN PAIR**

**30%**
**PAR LES TESTS**

**ÉTUDES MENÉES SUR UN PANEL DE 12.000 PROJETS**



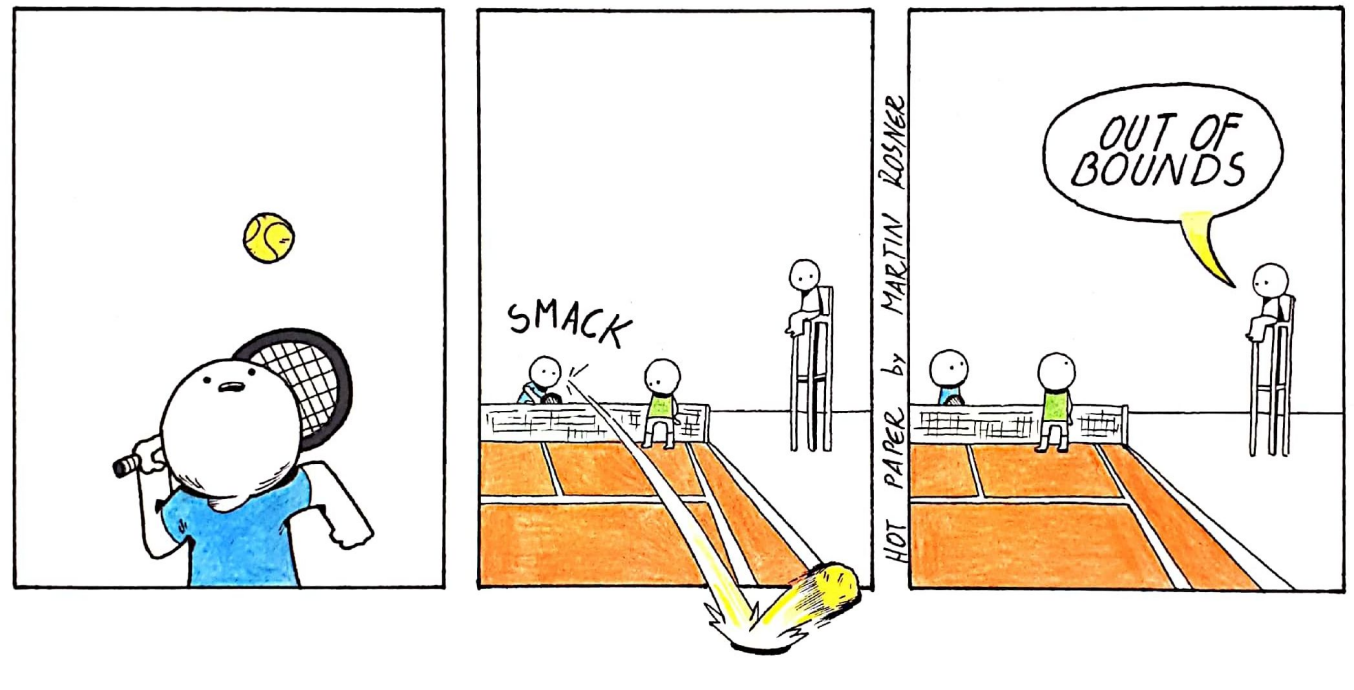
# Les différents types de revue de code

| | MOB PROGRAMMING | REVUE COLLECTIVE | REVUE PAR UN PAIR | PAIR PROGRAMMING |
|---|---|---|---|---|
| **EFFICIENCE** (nombre de défauts détectés) | +++ | +++ | ++ | ++ |
| **PROPRIÉTÉ COLLECTIVE DU CODE** | +++ | +++ | ++ | +++ |
| **AMÉLIORATION DE LA QUALITÉ, ÉVOLUTION DES STANDARDS** | +++ | +++ | +++ | +++ |
| **FACILITÉ DE MISE EN ŒUVRE** | + | + | +++ | ++ |
| **RAPIDITÉ DU FEEDBACK** | +++ | + | ++ | +++ |

# A vous de jouer !

# Revue de code sur le Kata Tennis Game

20 minutes




Emily Bache / Tennis Kata Refactoring : https://github.com/emilybache/Tennis-Refactoring-Kata

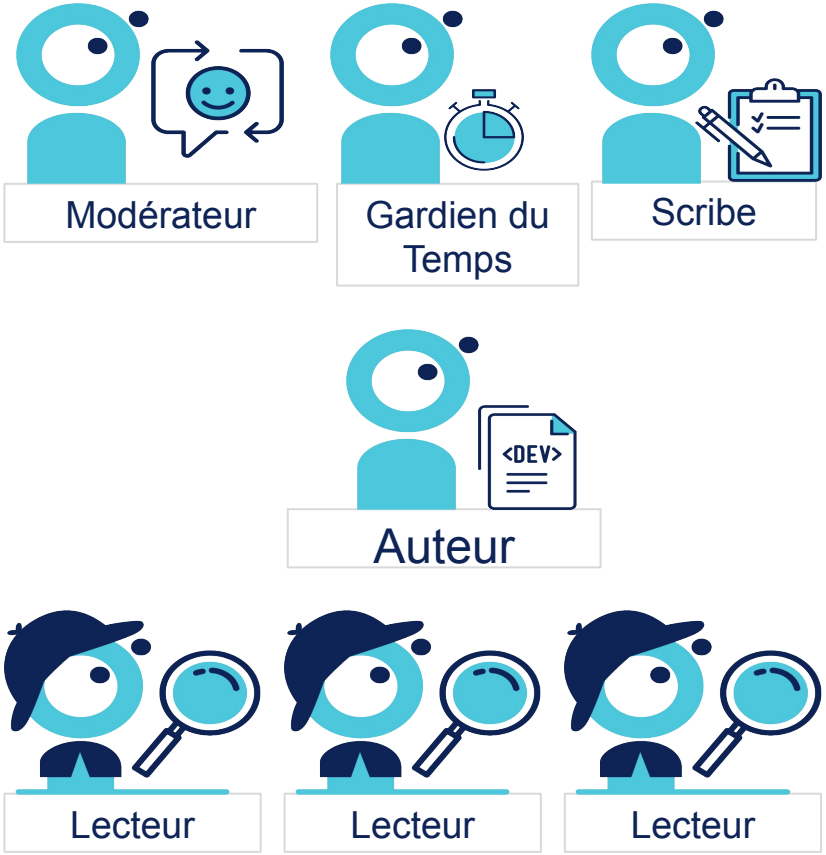
# Debrief : alors c'était comment ?

# Debrief de la revue de code

- Comment je me sens ?
- Qu'est-ce qui s'est bien passé ?
- Qu'est-ce qui s'est moins bien passé ?
- Qu'est-ce qui nous a manqué ?

# La revue de code : The Better Way

# Processus de la revue de code




**Documents généraux**

- liste des typos
- rapport de revue
- liste des points d'attention
- questionnaire débrief

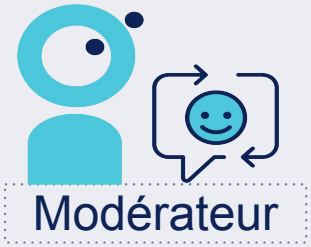
**Aides à la revue**

- checklist des défauts
- règles

**Métriques**

- données constatées
- reporting

# Processus de la revue de code - Préparation

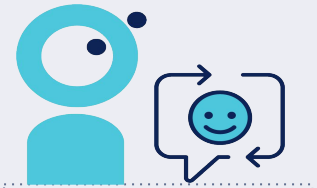



- Rappelle le RDV aux participants


- Transmet le code à relire aux participants en avance de phase


- Lisent le code et notent leurs remarques en avance de phase

# Processus de la revue de code - Déroulement



Modérateur

- Facilite la communication

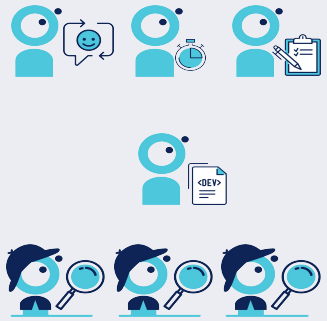
Scribe

- Note les points d'attention, les défauts
- Annote le code (outils…)
- Les typos (formatage…) sont ajoutés à la liste des typos

Gardien du Temps

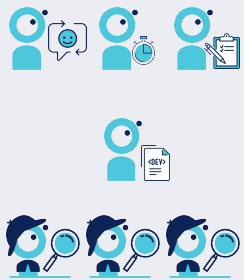
- Surveille le temps
- 1 minute par point

# Processus de la revue de code - Déroulement




- Tout le monde surveille son langage !

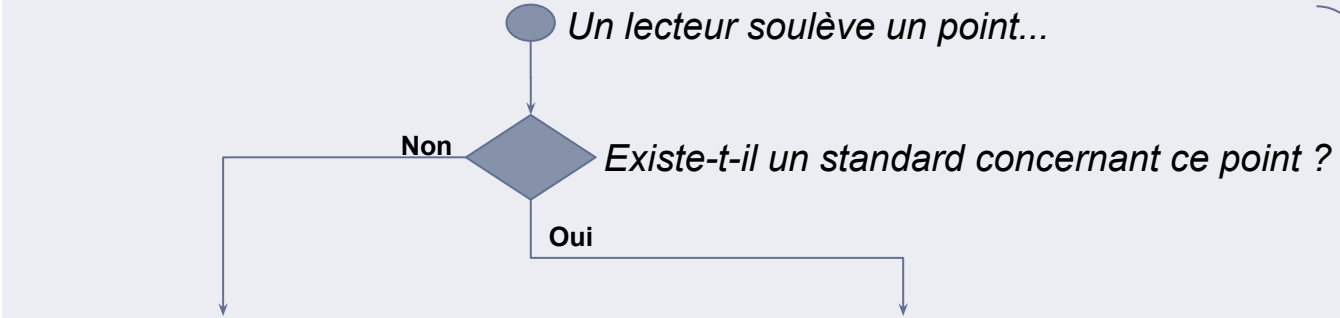
# Processus de la revue de code - Clôture et Suivi



- Par consensus, les participants statuent sur la revue:
  - Le code est approuvé
  - Le code est approuvé sous réserve de correction des points d'attention
  - A revoir : le code présente des anomalies importantes
  - La revue est incomplète


- Collecte toutes les données nécessaires
- Dresse et transmet un rapport de revue

# Objectif zéro débat !




L'amélioration du design et/ou la création d'un standard concernant ce point seront discutés hors du cadre de la présente revue de code.

On applique le standard. Si celui-ci continue de faire débat, on discutera l'éventuelle évolution de ce standard hors du cadre de la présente revue de code.

Le gardien du temps et le modérateur s'assurent que ce processus ne prenne pas plus d'une minute par point.

**> Éviter que la revue ne se transforme en débat d'opinions, c'est maximiser sa valeur pour l'ensemble des participants (personne ne s'ennuie, davantage de lignes sont revues) et assurer ainsi sa pérennité.**

# Objectif zéro débat !


Attention ces débats doivent avoir lieu !

**> Éviter que la revue ne se transforme en débat d'opinions c'est maximiser sa valeur pour l'ensemble des participants (personne ne s'ennuie, davantage de lignes sont revues) et assurer ainsi sa pérennité.**

# Pour une revue efficace

- Relire moins de 200-400 LOC par revue
- Viser un taux de revue de 300 LOC / heure
- Pas plus de 90 minutes de revue
- L'auteur annote son code avant la revue
- Se donner des objectifs et mesurer la revue afin d'améliorer le process
- Une checklist améliore les résultats pour l'auteur comme pour les relecteurs
- Les managers doivent encourager une politique de revue dans laquelle les erreurs sont vue positivement
- Attention à l'effet « Big Brother »
- L'effet « ego » : faites quelques relectures, même si vous ne relirez pas tout le code
- Des revues même légères sont efficaces et pratiques pour trouver des bugs
- Attention aux PR par commentaire interposé (Github, Gitlab, etc.)

# Les standards d'équipe

- De ces débats vont naître les standards de l'équipe !
- Les standards sont tracés et documentés (wiki, readme, ADR, …)
  - Pour pouvoir y faire référence
  - Pour que les nouveaux arrivants puissent s'y référer
- Les standards peuvent évoluer à travers le temps (mais il faut parfois faire du refactoring…)
- Exemple :
  - Choix de la langue
  - Type d'indentation
  - Règles de nommage
  - Glossaire

# La checklist

- La checklist est une pratique qu'on trouve notamment chez les pilotes d'avion
- Elle nous permet de vérifier qu'on n'oublie aucun contrôle dans notre code
- Elle se construit avec le temps et est personnel
  - > Même si une base peut être reprise d'internet ou construite collectivement par l'équipe
  - > Elle est souvent lié aux standards de l'équipe et parfois à l'architecture du projet
- Elle peut être difficilement exhaustive : focalisez vous sur les 10 ou 20 points les plus importants pour vous maintenant et dans votre projet



# Construisons notre check list !

-

# Et la revue par un pair ?

-

# Processus de la revue par un pair via Pull Request (PR)

# Pour des revues par un pair efficaces via PR

- Une branche de travail ne doit passer plus de 2 jours ouvrés sans revue de code
- Même si la PR est annotée dans un outil, chaque remarque doit être explicitée par un **échange oral** puis **hypothétiquement standardisé**
- Tout point sujet à débat doit être débattu **avec l'ensemble de l'équipe**
- Une checklist améliore les résultats pour l'auteur comme pour les relecteurs
- Vérifier que les défauts sont bien corrigés!
- Exécuter le code en local (pas seulement les tests automatisés mais aussi le fonctionnement global, notamment si vous n'avez pas de smoke tests)

# Round 2 : Review !

# Revue de code sur le Kata Gilded Rose

20 minutes

Emily Bache / Gilded Rose Refactoring : https://github.com/emilybache/GildedRose-Refactoring-Kata

# Gilded Rose

Hi and welcome to team Gilded Rose. As you know, we are a small inn with a prime location in a prominent city ran by a friendly innkeeper named Allison. We also buy and sell only the finest goods. Unfortunately, our goods are constantly degrading in quality as they approach their sell by date. We have a system in place that updates our inventory for us. It was developed by a no-nonsense type named Leeroy, who has moved on to new adventures. Your task is to add the new feature to our system so that we can begin selling a new category of items. First an introduction to our system:

- All items have a SellIn value which denotes the number of days we have to sell the item
- All items have a Quality value which denotes how valuable the item is
- At the end of each day our system lowers both values for every item

Pretty simple, right? Well this is where it gets interesting...

# Gilded Rose

- Once the sell by date has passed, Quality degrades twice as fast
- The Quality of an item is never negative
- “Aged Brie” actually increases in Quality the older it gets
- The Quality of an item is never more than 50
- “Sulfuras”, being a legendary item, never has to be sold or decreases in Quality
- “Backstage passes”, like aged brie, increases in Quality as it’s SellIn value approaches; Quality increases by 2 when there are 10 days or less and by 3 when there are 5 days or less but Quality drops to 0 after the concert
- “Conjured” items degrade in Quality twice as fast as normal items

# Debrief : alors, bien ou bien ?

# Debrief de la revue de code : Round 2

- Comment je me sens ?
- Qu'est-ce qui s'est mieux passé ?
- Qu'est-ce qui reste compliqué ?



# Propriété collective du code et communication

# Revue de code et attitude

**Programmation sans Ego** *(egoless programming)* : un ensemble de principes issus du livre de J. Weinberg: The Psychology of Computer Programming

- Comprends et acceptes que tu feras des erreurs
- Tu n'es pas ton code
- Même si tu connais beaucoup de "prises", il y a quelqu'un de meilleur que toi
- Ne réécris pas un code sans consultation préalable
- Traite ceux qui en savent moins que toi avec respect déférence et patience
- La seule chose constante dans le monde, c'est le changement
- La seule véritable autorité émane de la connaissance, pas du pouvoir
- Bats toi pour ce que tu crois, mais accepte gracieusement une défaite
- Ne t'isole pas au fond de la pièce
- Critique le code, pas la personne, soit gentil avec les développeurs, pas avec le code

# Comprends-le et accepte-le: tu feras des erreurs

Ce qui compte, c'est de les trouver au plus tôt, avant qu'elles n'arrivent jusqu'en production.

Heureusement, à part pour le peu d'entre nous qui développons du logiciel de guidage des fusées au CNES, les erreurs sont rarement fatales dans notre industrie, alors nous pouvons, et devrions, apprendre d'elles, en rire, et avancer.

# Tu n’es pas ton code

Rappelle toi que tout l’intérêt d’une revue est de trouver les problèmes, et des problèmes on va en trouver.

Quand un problème est découvert, **ne le prends pas personnellement.**

# Egoless Programming - 3

## Même si tu connais beaucoup de « prises », il y aura toujours quelqu'un de plus fort que toi

Ce quelqu'un peut t'apprendre de nouveaux trucs si tu le lui demande.

Recherche et accepte des retours de la part des autres, surtout si tu crois que ce n'est pas nécessaire.

# Ne réécris pas de code sans consulter quelqu'un

Il y a une frontière subtile entre « réparer du code » et « réécrire du code ».

Apprends à faire la différence, et chercher à améliorer le style dans le cadre des revues de code, et non pas comme un justicier solitaire.

# Egoless Programming - 5

## Traite les gens qui en savent moins que toi avec respect, déférence et patience

Les non-techniciens qui ont régulièrement affaire avec les développeurs ont presque tous l'opinion que nous sommes au mieux, des divas, au pire des bébés grincheux.

Ne renforce pas ces clichés par ta mauvaise humeur ou ton impatience.

# Egoless Programming - 6

## La seule constante dans le monde, c'est le changement

Sois ouvert au changement et accepte le avec le sourire.

Regarde chaque changement dans les exigences, la plateforme, ou les outils comme un nouveau défi, et non comme un dérangement majeur qu'il faudrait empêcher.

# Egoless Programming - 7

## La seule véritable autorité émane du savoir, pas de la position

Le savoir crée de l'autorité, et l'autorité créée le respect – donc si tu veux être respecté dans un environnement sans ego, cultive ton savoir.

# Egoless Programming - 8

## Bats-toi pour ce en quoi tu crois, mais accepte une défaite avec grâce

Admets que quelque fois tes idées seront rejetées. Même s'il s'avère que tu avais raison, **ne te venge pas**, ne répète pas **« je vous l'avais dit »** trop souvent, et ne fais pas de ton idée une idée martyre, ou un cri de ralliement.

# Ne sois pas « la personne au fond de la pièce »

Ne sois pas la personne qui code dans un bureau sombre et qui ne sort que pour acheter un coca. La personne au fond de la pièce est hors de portée, hors de vue, et hors de contrôle; elle n'a pas sa place dans un environnement ouvert et collaboratif.

# Egoless Programming - 10

## Critique le code pas la personne, soit doux avec les gens et dur avec le code.

Autant que possible, fais des remarques positives, allant dans le sens d'une amélioration du code, ayant à voir avec les standards locaux, les spécifications, de meilleures performances, etc.



# Comment rater sa revue de code ??

# Comment rater sa revue de code ?

- L'équipe ne progresse pas
- Les revues se transforment en débats sans fin
- Malgré la mise en place des revues, on croule toujours sous les bugs
- **Le tech lead passe tout son temps à faire les revues**
- Ca fait mal quand on critique mon code
- **Je n'ai pas de soutien de mon management**

# L’équipe ne progresse pas

- La revue n’occasionne pas d’échange
- L’auteur ne corrige pas les défauts
- Des standards non partagés

# Les revues se transforment en débats sans fin

- Il n'y pas de standards
- Il y a un désaccord sur les standards
- Les points « non standards » sont discutés lors de la revue

# Ca fait mal quand on critique mon code

- Les revues de code font l'objet de « **finger pointing** »
- J'ai peur d'être mis en défaut
- Les revues de code manquent de **bienveillance**

# Malgré la mise en place de revues, on croule toujours sous les bugs

- La revue n'a pas été préparée
- La quantité de code à revoir est trop élevée
- On ne sait pas quels défauts relever
- Les défauts détectés ne sont pas corrigés

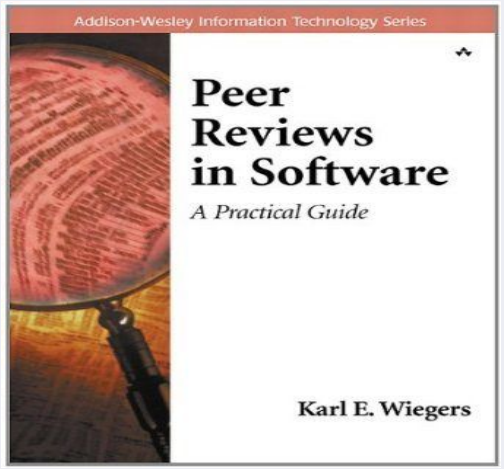
# Je n’ai pas le soutien de mon management

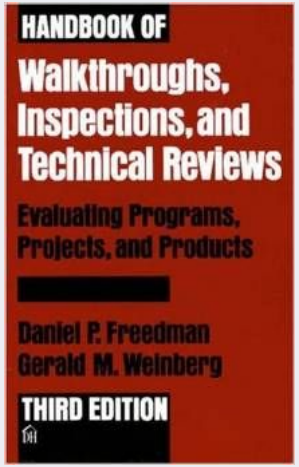
- Le management dit que c’est inutile
- Le management me dit que je n’ai pas le temps
- Le management se sert des revues de code pour fliquer les développeurs

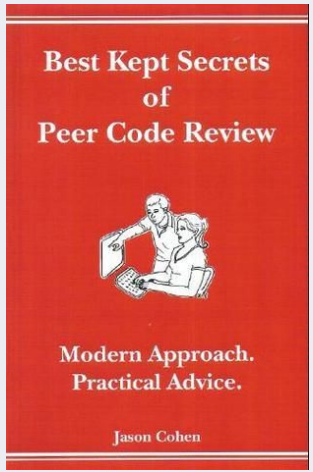
# Le Tech Lead passe tout son temps à faire des revues

- Seul le tech lead connait le code
- Seul le tech lead est suffisamment expert
- Le tech lead n'a pas confiance dans les autres

# Références




http://smartbear.com/resources/whitepapers/best-kept-secrets-of-peer-code-review/

https://courses.utep.edu/portals/870/Lecture-16a-Walkthroughs.ppt

# REX ? Question ? Pourboire ? Jet de tomates virtuelles ?