La revue de code

Lionel DURAND

Avril 2023

lioneldurand@gmail.com
https://www.linkedin.com/in/lionel-durand-649b72193/

Pourquoi faire des revues ?

Pourquoi faire des revues ? 1/3

Pour produire un meilleur code

- qui fait ce qu'on attend de lui
  - valider le fonctionnel
  - trouver des bugs
  - trouver des bugs le plus tôt possible
- qui est maintenable
  - testable et testé
  - simple à comprendre

Pourquoi faire des revues ? 2/3

Pour améliorer la maitrise collective du code

- parler du code entre membres de l'équipe
- encourager la propriété collective du code
- partager les responsabilités
- partager les standards de dév
- faire évoluer les standards de dév

Pourquoi faire des revues ? 3/3

Pour apprendre

- montée en compétence des nouveaux arrivants
- l'auteur apprend des retours du relecteur
- le relecteur apprend du code de l'auteur

Quels types de revues ?

Comment et par qui ?

Pas de définition unique, pas de modalité unique

Il faut que le tech lead valide toutes les revues de code

Il faut au moins 1 relecteur en plus de l'auteur

Il faut qu'au moins 2 membres de l'équipe participent à l'écriture du code

Ici, tout est développé en mob programming

Qui valide la revue ?

Les architectes de l'entreprise	=> Oligarchie
Le tech lead de l'équipe	=> Autocratie
L'équipe	=> Démocratie => Démocratie participative (représentant tournants)
Toute l'équipe	=> Holacratie => Anarchie (personne ne décide)

Les types de revues courants

Ecriture puis relecture (asynchrone)

- Revue par un ou plusieurs pairs
- Revue "par dessus l'épaule"
- Revue formelle (durée, périmètre)
- Revue via un outil (ex gitlab)

Ecriture à plusieurs (feedbacl temps réel)

- Pair programming
- Mob programming

Les types de revues courants

	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	+	++	+++

Défauts courants et bonnes pratiques

La revue n'améliore pas la qualité du code

Symptômes

- Le nb de bugs ne diminue pas
- Le code reste peu lisible
- Notre vélocité diminue
- La dette technique augmente

La revue n'améliore pas la qualité du code

Bonnes pratiques

- qu'est ce que les revues doivent apporter à l'équipe ?
- travailler sur des standards de code
- les construire progressivement
    - traiter les plus grosses douleurs en premier
    - compléter les standards au fur et à mesure
- checklist de points d'attention prioritaires pour les revu
- alterner les types de revues, pour faire du pair programm du mob programming

Le process de revue prend trop de temps

Symptômes

- Multiples allers-retours entre l'auteur et le relecteur
- On s'attend
- Il faut rebaser plusieurs fois
- On n'arrive pas à prendre de décision
- On ne se comprend pas

Le process de revue prend trop de temps

Bonnes pratiques :

- plus le périmètre est petit, plus c'est efficace : 2 merge requests valent mieux qu'une !
- prioriser la relecture sur toutes les autres taches
- se prévenir pour ne pas s'attendre
- formaliser le processus de décision
    - à l'écrit, catégoriser les retours : [FIX], [OPT], [QU
    - à l'oral : time boxer, décider vite ou reporter
- être précis et univoque, demander si on n'est pas sur
- standards de code et check liste

La revue est mal vécue

Sympômes

- Peur d'être mis en défaut
- Sentiment de reproche
- Sentiment de ne pas être écouté
- Tensions

La revue est mal vécue

Bonnes pratiques

- soigner la communication
- apprendre à donner et recevoir des feedbacks
- dépersonnaliser : "le code produit un bug" plutôt que "
  bug"
- partager la responsabilité : "nous" plutôt que "tu" ou "j
- relecteur : faire aussi des retours sur ce qui est bien
- auteur : dire quand on se sent mal

Donner et recevoir du feedback

La CNV appliquée à la revue de code

- 1/ observation : ce que je vois
    - "cette fonction fait 30 lignes"
- 2/ inférence et impact : l'impact que j'imagine sur le cod
  l'équipe
    - "la prochaine fois qu'on va intervenir dessus, on
      mal à comprendre"
- 3/ demande éventuelle : ce que je propose
    - "peut-être qu'on pourrait la découper en plusieu
      fonctions"

Programmation sans ego

J. Weinberg, The Psychology of Computer Programming

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

Les merge requests

Revues en ligne

Selon l'outil : Merge request (gitlab), pull request (github), ou autre...

Principes similaires

- l'auteur demande la fusion de sa branche via l'outil
- le relecteur peut commenter
- l'auteur peut valider et merger

De plus en plus utilisé

Revue via des merge requests

Les plus :

- asynchrone avec process formel
- le relecteur fait ses remarques "dans le code"
- l'auteur voit les remarques sous forme de TODO liste
- intégration au workflow git & gestion des issues
- suivi

Via un outil : Les merge requests

Les points de vigilance :

- asynchrone : on risque de s'attendre
- remarques par écrit : favorise l'incompréhension et les tensions

=> mettre en oeuvre les bonnes pratiques vue précédemment

=> combiner avec d'autres modes de revue (synchrones)

Conclusion

Conclusion

Des gains pour le produit (qualité), pour l'équipe (collaboration), et pour ses membres (apprentissage)

Accepter de passer du temps à lire du code (plutôt que d'en écrire)

Formaliser (et faire vivre) un process de revue adapté au projet et à ses membres

Ressources

Livre blanc Culture Code de Octo