3.4 - (TODO) Relecture de code

3.4.1 - Présentation

3.4.2 - Mise en œuvre

3.4.2.1 - Développement

Le développement se fait sur une branche GIT séparée. Une fois le développement terminé (nous verrons ce que l'on met sous ce terme), le travail est prêt à passer à la phase de relecture.

NDRVE> La préco, c'est git-flow:

https://jeffkreeftmeijer.com/git-flow/

https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Pour cela, on crée une MergeRequest via l'interface gitlab.

Cette merge request peut être commentée et annotée. C'est un moyen de donner au relecteur de informations sur la procédure de test, si des particularités existent (je pense notamment aux jeux de données).

Conflits

Cette Merge Request doit pouvoir être fusionnée automatiquement par GIT. Si ce n'est pas le cas, Gitlab vous le fera savoir. C'est le rôle de la développeuse ou de développeur de résoudre le conflit.

Il est aussi possible que la merge request soit fusionnable au moment de sa création, et ne l'est plus au moment de la relecture.

Pour cela, il faut fusionner la branche principale dans la branche en cours, résoudre les conflits et pousser le commit.

Si la branche de développement est assez vieille, il peut aussi être intéressant de récupérer les développement de la branche principale en amont de la merge request.

Développement terminé ?

Avant de passer le code en relecture, il y a quelques pré-requis évidents :

- Le code compile
- La fonctionnalité remplit bien son but (ie, le développeur l'a testée)
- Des tests unitaires existent pour la fonctionnalité
- Tous les tests unitaires existant passent
- La couverture de code n'est pas dégradée
- Sonar ne remonte pas de nouvelle alerte

- Les méthodes/classes publiques sont documentées, ainsi que les détails spécifiques d'implémentation qui ne sont pas évidents au premier abord
- Commenter son code pour la postérité. Penser au dev qui va le relire dans 2 ans.

Pour le confort du relecteur :

- Code lisible et bien indenté
- Ne pas ré-indenter le code non modifié (cela crée des 'fausses' différences, pénibles à trier pour le relecteur)

Libre à chaque projet d'établir ses propres règles.

3.4.2.2 - Relecture

Quand : Le plus tôt est le mieux, pour raccourcir au plus le délai développement - validation

1. S'assigner la merge request sous gitlab, afin d'éviter que deux personnes relisent la même branche
2. Relire les changements, discuter avec le développeur...
3. Tester le développement :
    1. récupérer la branche en local : git fetch, git checkout feature/nnn
    2. compiler, vérifier les TU, déployer, tester

3.4.2.3 - Fusion

La fusion peut s'opérer soit via gitlab, soit sur le client.

Via gitlab :

1. Changer le message du commit pour faire apparaître en première ligne l'identifiant du développement (feature xxx, redmine nnn). Cela facilite la lecture du graphe des commits.
2. Cocher la case 'Remove source branch'.
3. Fusion

Via le client git :

1. Mettre à jour son référentiel local : git pull (sur la branche cible et sur la branche source)
2. Se placer sur la branche cible (git checkout develop, git checkout release/XX ...)
3. Lancer le merge :
    1. git merge --no-ff feature/xx

        L'argument --no-ff, no-fast-forward, permet de forcer la création d'un commit, même si un fast-forward était possible. Il y a donc une saisie d'un message de commit qui s'ensuit.

        Préciser dans le message le but du commit en première ligne (feature xxx, hotfix yyy). Toujours pour des raisons de lisibilité ultérieure.
4. Pousser le commit : git push
5. Supprimer la branche source distante : git push --delete origin feature/xxx

    Et la branche locale : git branch -d feature/xxx

3.4.2.4 - Déploiement

Votre job jenkins préféré devrait prendre en compte votre merge sous peu et le rendre disponible au recetteur pour validation.

Suivant le projet et sa prise en compte, il peut aussi y avoir des opérations manuelles à effectuer (SQL à jouer sur la validation, lib à ajouter, etc).

Ne pas oublier de les lancer.

3.4.2.5 - Ensuite ?

C'est rare, mais ça arrive :

Il est possible que la recette remonte une erreur sur le développement effectué.

Il faut dans ce cas recréer une branche à partir de la branche de dev à jour. Ne pas repartir de l'ancienne branche (qui a été supprimée).

Le même couple developpeur/relecteur opère à la correction.

En bonus, vous pouvez aussi ouvrir un débat sur pourquoi l'erreur est passée au travers du système, et est-il possible de faire mieux ?

3.4.2.6 - Nettoyage GIT

À l'usage, on se retrouve vite avec plein de branches en local qui n'ont plus d'existence sur le repository distant.

C'est gênant dans le cas d'une remontée d'erreur par le recetteur. Vous allez vouloir recréer une branche avec le même nom, ça va poser problème, à vous et potentiellement aux autres.

Malheureusement, il ne semble pas y avoir de recettes miracle. La meilleure solution que j'ai trouvé pour le moment consiste à lister les branches, repérer celle flaguée 'gone' et les supprimer :

```
git fetch -p ; git branch -vv | grep ': gone]' | awk '{print $1}' | xargs git branch -d
```