SATENSKE

Zeste de savoir

La revue de code, c'est important !

21 janvier 2019

Table des matières

1. Introduction

Si vous avez déjà travaillé à plusieurs sur un projet logiciel plus ou moins conséquent, vous avez certainement dû être confronté à au moins un de ces problèmes :

— une dette technique difficilement rattrapable ;
— du code incompréhensible ;
— des *bugs* qui auraient dû être vus avant la livraison ;
— toute la connaissance technique du projet portée par une seule personne.

Une des solutions pour limiter ce genre de problème, c'est la revue de code. Voyons un peu son intérêt et ce que ça peut apporter dans un projet logiciel.

> *i* Je parle de limiter les problèmes, la revue de code ne s'oppose pas à des tests automatisés ou à une bonne gestion de projet bien entendu.

2. La revue de code

Pour commencer, une petite définition de la revue de code tirée de Wikipédia :

> La **revue de code** (de l'anglais *code review*) est un examen systématique du code source d'un logiciel.
> Il peut être comparé au processus ayant lieu dans un comité de lecture , l'objectif étant de trouver des bugs ou des vulnérabilités potentielles ou de corriger des erreurs de conception afin d'améliorer la qualité, la maintenabilité et la sécurité du logiciel.
>
> *https ://fr.wikipedia.org/wiki/Revue_de_code*

Le principe est assez simple : un autre regard sur notre travail est assez bénéfique. Chaque développeur, et plus généralement chaque personne, a une façon différente de penser et d'appréhender un problème. Un autre regard peut donc amener une autre approche.

La revue de code se fait de plus en plus régulièrement dans des entreprises de différentes tailles, aussi bien dans des petites *startups* de quelques développeurs que dans des mastodontes tel que Google ou le projet Linux .

Une revue fait donc intervenir au moins deux personnes, une personne qui soumet le code, que j'appellerai « l'auteur », et une ou plusieurs personnes qui relisent le code, le « relecteur ».

2.1. Qui peut relire le code

Il y a deux écoles, les deux ont pour moi leurs avantages et leurs inconvénients.

Un développeur expérimenté, tel que le *lead dev* va relire l'ensemble du code de l'équipe. Ce développeur ayant la meilleure connaissance de l'application et du métier, il sera plus à même de relire le code de l'équipe. L'inconvénient est que la connaissance de l'application restera au sein de la même personne, si un jour ce développeur est absent ce sera dommageable pour le projet.

Il est également possible de faire tourner le relecteur, chaque personne sera amenée à lire le code d'un autre développeur. Le gros avantage est de responsabiliser le développeur, et de faire monter en compétence l'ensemble de l'équipe, cela permet également de partager la connaissance du projet. Toutefois, cette solution peut être plus difficile à mettre en place. En effet, si le projet est conséquent, chaque développeur n'a pas toujours une vision assez globale du projet pour relire efficacement le code de quelqu'un d'autre. De plus, la relecture de code n'est pas quelque chose de simple, un bon développeur peut avoir du mal à se plonger dans le code de quelqu'un d'autre. Il sera donc nécessaire de faire monter l'équipe en compétence.

Ces deux approches ne sont pas forcément exclusives. Il est aussi possible d'avoir d'abord une relecture par un autre développeur, et que ça soit le *lead dev* qui valide l'intégration. Cela permet de combiner les deux avantages, en revanche, cela va prendre plus de temps, en raison d'une double validation.

2.2. Comment faire une revue de code

Il y a plusieurs manières d'aborder une revue de code. Celle-ci dépend principalement du contexte du projet et de la culture de l'entreprise. Lorsque le développeur estime avoir terminé le changement, il peut demander une relecture. Voici plusieurs manières pour effectuer cette relecture :

— Les *pull* ou *merge requests* : ce sont des méthodes très utilisées dans le monde du libre notamment au travers de Github ou Gitlab . Un développeur envoie ses changements et une autre personne valide et intègre le moment venu.
— « Analyse par-dessus l'épaule » : deux personnes sont devant l'ordinateur, l'auteur commente son code et la seconde personne donne éventuellement des conseils ou pose des questions.
— La revue de code par *mail* : c'est une technique qui avait cours avant la démocratisation des outils plus modernes tel que Github, il est possible de faire une simple liste de diffusion et de s'envoyer des *patchs* par *mail*. La relecture se faisant directement par retours de *mails*.

3. Pourquoi faire de la revue de code ?

À ce stade, vous êtes peut-être en train de vous dire que ça semble une bonne idée, mais que ça va prendre beaucoup de temps et d'énergie. Je ne peux pas vous contredire sur ce point. Bien que le temps de la revue soit variable d'un développeur à l'autre, oui, ça peut être long. Mais je pense que le temps passé est contrebalancé par les avantages. À long terme, cela permettra d'avoir un projet plus maintenable, avec une équipe efficace.

3.1. On écrit du code lisible et propre

Quand un projet se met à la revue de code, c'est souvent le premier objectif : avoir un code plus lisible et globalement de meilleure qualité. Et en effet, ça marche.

L'auteur va s'obliger à relire son propre code, il va donc ajouter des commentaires et tout faire pour qu'il soit le plus compréhensible possible pour que la revue de code se passe bien.

Si le code est intégré, cela veut dire qu'au moins deux personnes ont estimé que le code était lisible et élégant. On peut extrapoler un peu et se dire que si deux personnes de l'équipe le valident, le reste de l'équipe devrait également le comprendre sans difficulté.

Par ailleurs, un second regard à l'application permet de limiter la duplication de code grâce à une deuxième vision. « J'avais codé *ça* pour un autre module, on doit pouvoir le réutiliser plutôt que réinventer la roue ! ».

3.2. On limite les *bugs*

Le relecteur ne va pas forcément tester le code, mais va avoir un œil extérieur, contrairement à l'auteur qui était plongé dans son code. Le relecteur peut ainsi:

— identifier certains *bugs* en lisant le code, tel que des cas limites que le développeur aurait pu oublierou des problèmes d'implémentation ;
— constater une mauvaise compréhension du besoin à la source du changement ;

Le relecteur peut suggérer l'ajout de tests unitaires pour des cas qui n'auraient pas été prévus initialement. C'est un bon moyen de vérifier que le patch est fonctionnel, et que les cas limites sont correctement gérés, tout en améliorant la qualité des tests.

3.3. On apprend

Ce n'est pas forcément la première chose à laquelle on pense quand on parle de revue de code, mais de mon expérience la revue de code permet d'apprendre. Que ce soit pour l'auteur ou pour le relecteur (principalement dans le cas où les relecteurs « tournent »).

— Le relecteur peut tomber sur quelque chose d'inconnu : « oh, tiens ! Je ne connais pas ça ? Comment ça marche ? » L'auteur va ainsi expliquer ce qu'il a fait et en quoi il estime que c'est avantageux.
— Mais on peut avoir l'effet inverse : « tu pourrais utiliser cette technique, c'est plus optimisé / élégant / efficace ! »

On peut aussi apprendre sur des questions de métiers ou des questions d'architecture de notre application. Ces échanges sont toujours bénéfiques.

3.4. On enrichit la documentation

La revue de code est le meilleur moment pour poser des questions. Lorsqu'un non-expert est relecteur, il peut poser des questions sur un aspect qu'il ne comprend pas. Cela ne veut pas forcément dire qu'il y a un problème. Par contre, cela veut dire qu'il y a du passage de connaissance à effectuer.

C'est une bonne occasion pour améliorer l'existant. Cela peut venir de commentaires manquants ou plus globalement d'un problème de documentation. C'est alors le bon moment pour améliorer cette documentation, ce qui pourra ensuite profiter à toute l'équipe.

3.5. On communique

Ça peut paraître assez basique, mais le simple fait d'avoir une revue de code va obliger l'équipe à communiquer, expliquer ce que chacun a fait, comment et justifier les différents choix.

Une revue de code est l'occasion rêvée pour poser des questions, y répondre et discuter.

3.6. On peut former les nouveaux

La revue de code est déjà souvent utilisée pour les nouveaux développeurs, mais avoir une revue de code systématique pour l'ensemble de l'équipe permet au nouveau de se sentir complètement intégré à l'équipe : il est dans les mêmes *process*.

Elle est d'autant plus importante pour lui, qu'il ne connaît ni l'application, ni le métier. C'est une bonne occasion pour lui faire des remarques, lui expliquer des choses, lui montrer les bonnes méthodes limitant la duplication de code. D'une entreprise à l'autre, les approches peuvent être différentes, il est donc important de lui transmettre les habitudes de l'équipe.

Une fois qu'il aura bien appréhendé le projet, le fait de relire le code de développeurs plus expérimentés lui permettra d'apprendre de nouvelles choses et de prendre confiance en soi.

4. Faire une bonne revue de code

On va essayer de voir de quelle manière il est possible de mettre en place de la revue de code dans un projet, et, surtout, de faire en sorte qu'elle se passe le mieux possible.

4.1. Les outils

Il est très utile de s'outiller pour effectuer de bonnes revues de code. C'est nécessaire principalement pour gagner du temps, mais aussi pour améliorer la qualité des revues.

4.1.1. La revue de code : Github, Gitlab et Bitbucket

La revue de code est globalement plus simple avec un système de versionnement moderne tel que Git, car ils permettent de créer facilement une branche, il est alors aisé de faire relire notre branche. Les trois outils que je vais vous présenter sont les outils les plus connus et utilisés avec Git.

> Pour d'autres systèmes tel que SVN, CVS, Mercurial, bazaar etc. vous pouvez regarder du côté de Phabricator , Review board ou Crucible .
> Il en existe d'autres mais je ne les connais pas, un peu de recherche et vous devriez trouver votre bonheur.

Ces outils permettent principalement d'avoir un fonctionnement *asynchrone* : le développeur propose son code et le relecteur peut le relire un peu plus tard. Il n'est pas nécessaire d'avoir les deux personnes disponibles en même temps, comme ça peut-être le cas avec la méthode « d'analyse par-dessus l'épaule » par exemple.

Ces trois outils sont assez similaires en termes d'utilisation. La principale différence vient du coût et du *business model* :

- Github , très connu et utilisé dans le monde du libre. Il n'est gratuit que si le projet est *open source*, sinon il faut payer pour un nombre de dépôt fermé.
- Gitlab , est un projet libre similaire à Github, il permet d'installer une version communautaire. Cela demandera un peu d'administration système et un serveur à disposition. L'entreprise Gitlab propose également une version *Entreprise Edition* permettant d'utiliser leurs serveurs.
- Bitbucket fait partie de la suite Atlassian , il a donc une bonne intégration avec leurs outils tel que Jira .

Le développeur va coder sa fonctionnalité ou corriger son *bug* sur une branche. Une fois qu'il estime avoir terminé son travail et qu'il souhaite une relecture, il peut ouvrir une *pull request* (Github et Bitbucket) ou une *merge request* (Gitlab).

À ce moment-là, un sujet de discussion est ouvert contenant la différence de code ainsi qu'un espace dédié aux commentaires. Il est alors possible d'ajouter un commentaire dans le code ou de mettre des commentaires plus généraux.

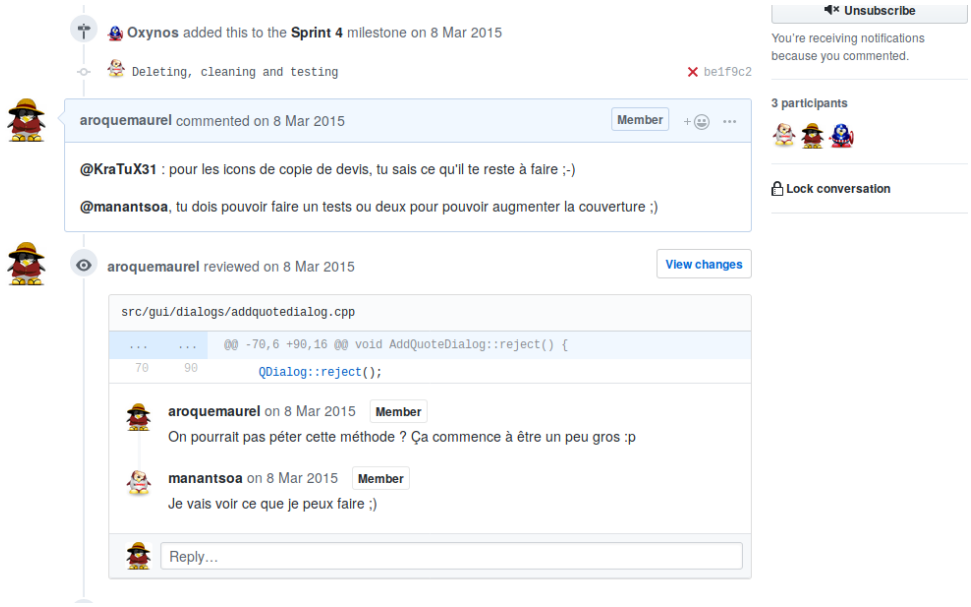

FIGURE 4. – Exemple de pull request sur Github

4.1.2. Les outils d'intégration continue

Il est possible de lier l'outil de revue de code à un outil d'intégration continue. Il faut voir ça comme un complément à une bonne revue. Des outils permettent d'effectuer une vérification automatique et un humain va faire une revue permettant d'améliorer significativement la qualité du projet comme nous l'avons vu précédemment.

Cet outil peut effectuer plusieurs actions, telles que :

- Compiler le projet.
- Lancer des tests automatisés (unitaires, de non-régression, de performance, ...).
- Calculer le taux de couverture des tests.
- Effectuer une analyse statique, l'idée est de faire une « relecture » du code par un outil qui va appliquer une « *checklist* » interne, afin de voir en amont certains *bugs* possibles ou de non-respects de conventions de codage.[1]
- Générer la documentation technique

Il existe une multitude d'outils d'intégration continue. Les plus connus sont Jenkins , Gitlab-CI et Travis-CI . Ceux-ci sont capables de mettre des commentaires directement dans la revue de code.

4.2. Une bonne revue de code, c'est un effort collectif

Pour une bonne revue de code, il faut être en bonne condition. Aussi bien du côté de l'auteur que du relecteur.

1. Pour l'analyse statique, l'outil dépend de la technologie utilisée. Un des plus connus multi-langage est SonarLint .

4.2.1. Faire des petites revues

Pour que ça se passe bien pour tout le monde, et surtout pour qu'elles soient les plus efficaces possibles, il faut faire des petites revues. Il n'est agréable pour personne d'avoir 10 000 lignes de code à relire.

— L'auteur a envie que son travail soit intégré rapidement et sera réfractaire aux modifications.
— Le relecteur n'a pas envie de tout relire, il va aller au plus vite.

Tout le monde est globalement épuisé. La relecture ne sera pas efficace. Il faut essayer de faire des petites tâches qui sont intégrées sur une branche de développement au fur et à mesure.

4.2.2. Anticiper les revues sur le planning

Si une *deadline* est proche, la revue de code va être précipitée. Elle risque d'être bâclée et si des modifications ou du *refactoring* doivent avoir lieu, ils risquent de ne pas être fait et de provoquer de la dette technique. Globalement, une revue de code, ça se prévoit, il est donc préférable de prendre en compte le temps de relecture d'une part, et les aller-retours d'autre part.

4.2.3. Être dans le bon état d'esprit

Ici, il s'agit d'un problème humain : il faut accepter que son travail soit analysé et éventuellement critiqué.

— Que ça soit du côté de l'auteur, qui doit accepter les critiques et essayer de les prendre en compte si possible.
— Ou du relecteur, qui ne doit pas avoir peur de donner des remarques sur le travail de son collègue. Mais surtout, il doit argumenter son avis et présenter les choses de manière à tirer le développeur vers le haut. Il est intéressant de donner des remarques positives et négatives. Cela peut permettre de motiver l'auteur et d'éviter qu'il ne se concentre que sur le négatif.

Une bonne revue doit susciter des échanges, des discussions, des remises en question.

De manière générale, que ça soit l'auteur ou le relecteur, les deux personnes doivent avoir une attitude positive.

FIGURE 4. – Commit Strip sur la revue de code

5. Conclusion

Merci d'avoir lu jusqu'au bout ! Il ne reste plus qu'à essayer de mettre en place de la revue de code dans vos projets.

Je sais que j'ai présenté un monde idéal : on revoit systématiquement l'ensemble du code. Ce n'est pas toujours évident, surtout pour les *bug fix* qui doivent être corrigés rapidement. Cependant, il est toujours possible de commencer à faire un peu de revue sur les nouveaux développements, et essayer d'étendre le périmètre vers les corrections de *bugs*, pour finalement arriver à 100% de revue de code !

Et surtout, n'oubliez pas :

— restez positifs ;
— faites de petites revues ;
— prévoyez les revues ;
— les outils peuvent effectuer un travail complémentaire.