Les 12 Bonnes Pratiques De l'Xtreme Programming

Client sur site:

Le client, un ami, m'a régulièrement envoyé ses attentes via les réseaux sociaux.

Jeu du planning ou planning poker

Au préalable le travail à été découpé en *fonctionnalités* dans un tableau trello. Si on ajoute le fait de développer seul, il est aisé de faire un point quotidien (à la livraison d'une fonctionnalité).

Intégration continue

Utilisation de travis-ci pour l'intégration continue. Quand je push un commit sur github, les tests sont effectué sur travis-ci puis le serveur est mis à jour.

Petites livraisons

L'intégration continue et les tests réduisent considérablement le coût de livraison.

Rythme soutenable

L'équipe ne fait pas d'heures supplémentaires.

Tests unitaires/fonctionnels

Avant de mettre en œuvre une fonctionnalité, j'écris un test qui vérifiera que mon programme se comporte comme prévu. Ce test sera conservé jusqu'à la fin du projet, tant que la fonctionnalité est requise. À chaque modification du code, je relance tous les tests que j'ai écris.

Conception simple

Conception simple et intuitive du projet.

Refactoring (ou remaniement du code)

Une fois le prototype du code construit et fonctionnelle, j'ai pu commencer le refactoring.

Appropriation collective du code

Développement effectué seul.

Convention de nommage

Python d'une part et *Django* ensuite sont deux cadre de travails qui prône les conventions de nommage avant le codage.

Pair-programming

Développement effectué seul.