Software craftsmanship

Le software craftsmanship est un mouvement qui a pris forme en 2008 avec la cinquième édition du **Manifeste agile**, « Craftsmanship over Execution », autrement dit, « l'artisanat plus que l'exécution ».

Le mouvement du software s'est inspiré des concepts développés dans le livre « ***The pragmatic programmer: from journey man to master*** » de Andy Hunt et Davis Thomas en 1999 ; un grand classique pour tous les développeurs et programmeurs du monde entier.

Le mouvement du software craftsmanship met en exergue le concept puissant qu'« **il ne suffit pas qu'un logiciel soit fonctionnel, il faut qu'il soit bien conçu** ». Le software **craftshiping** consiste donc à remettre en question certaines pratiques de développement. L'accent est mis sur la fiabilité et la facilité de maintenance, grâce au travail de développeurs performants.

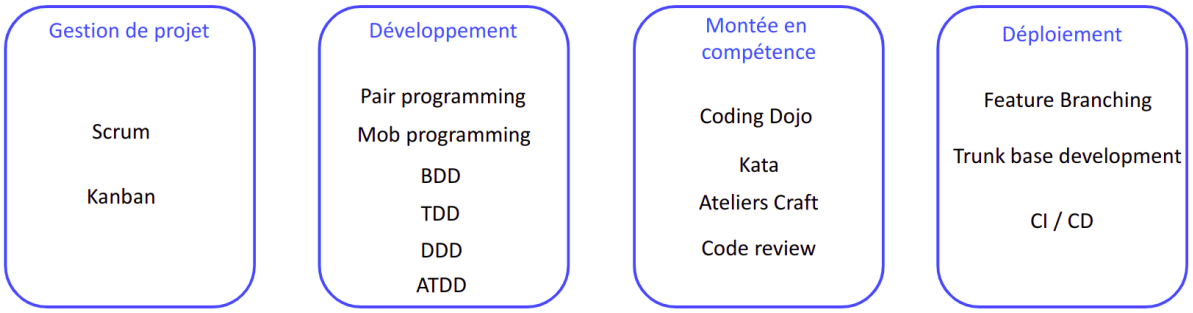
Software craftsmanship: clean code ?

Dans cette méthode est visé un haut niveau de qualité dans le **développement de logiciels**, dans une société où le logiciel est partout.

De nos jours on rencontre en entreprise des logiciels mal entretenus soit à cause d'une mauvaise architecture logicielle, Une dette technique élevée au niveau des applications dû à la pression de livraison, Manque de factorisation du code, présence de smell code etc.

Toutes ces choses sont dues à de mauvaise décision très souvent prise à l'origine du projet, il est important pour l'équipe de développement de mettre en pratique des conceptes comme : SOLID (**S**ingle responsibility principle, **O**pen close principle, **L**iskov principle , **I**nterface segregation principle, **D**ependency inversion principle), etc.

Voici quelques outils à mettre en place sur un projet.



Quelque notions pratiques à mettre en œuvre :

Le Formatage :

- Les variables doivent être déclarées au plus proche de leur utilisation,
- les membres de la classe doivent être déclarés en haut,
- Les fonctions liées doivent être proches. L'appelant avant l'appelé.

Le Nommage

- Le nom d'une classe, méthode ou variable devrait révéler son intention,
- Utiliser les noms longs,
- Utiliser les noms du domaine métier (Basket plutôt que ItemList),
- Utiliser les conventions entre développeur .

Fonctions

- Éviter les duplications de code,
- Un seul return par fonction,
- un seul niveau d'abstraction,
- une seule intention par fonction,

- le nom d’une fonction doit démarrer par un verbe
- Une fonction ne doit pas avoir trop de paramètre, un seul au maximum

La gestion des erreurs :

- Préférer la gestion des exceptions aux codes d’erreur,
- Utiliser les Try/Catch ,Ne pas mettre la gestion des exceptions dans la fonction qui effectue le traitement, Mais faire une fonction qui gère les exceptions et une autre fonction qui ne fait que le traitement.

Les classes

- Le principe Single Responsibility,
- Une seule raison devrait nécessiter le changement d’une classe,
- Une classe doit être responsable d’une seule préoccupation,
- Cohérence,
- Peu de variables,
- Des variables connectées aux fonctions,
- Couplage,
- Peu de connexion entre les classes,
- Préférer plusieurs petites classes qu’une seule grosse classe

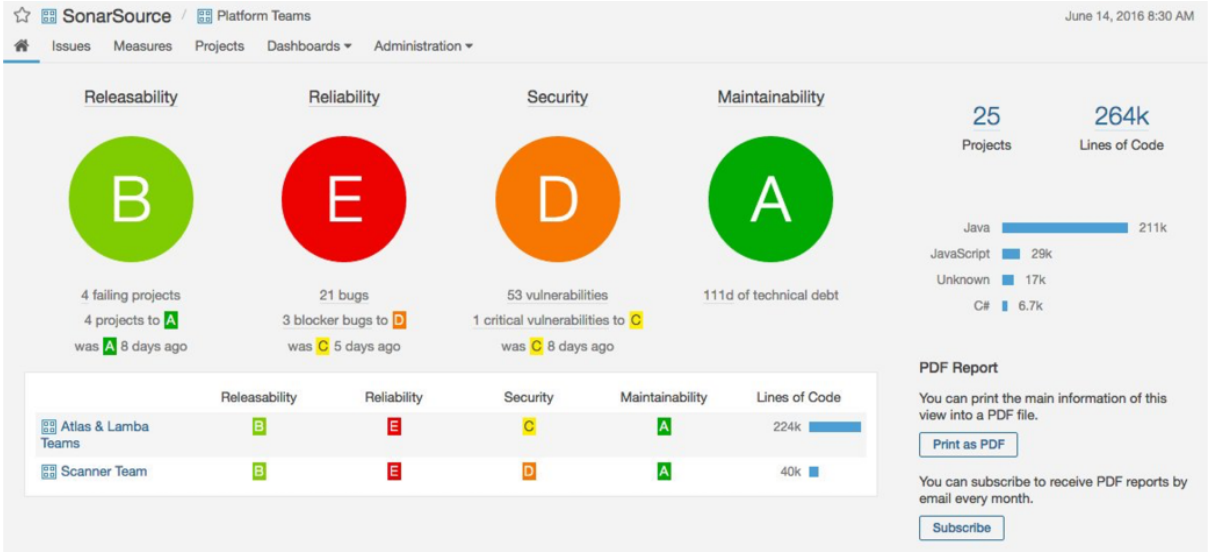
Les commentaires :

- En général, les commentaires sont une mauvaise pratique,
- Il faut les déplacer avec le code,
- Il faut les faire évoluer avec le code,
- Ce sont donc souvent des mensonges,

- Un concept et un nom bien choisis éliminent souvent le besoin d'un commentaire

Il existe des outils qui permette de mesurer la qualité du code à l'instar de sonar cube qui est un logiciel open source.

- Avec cet outil on peut : Respecter des règles et normes du code
- créer une documentation du code,
- Analyser des tests unitaires (couverture du code, etc.)
- Dupliquer du code
- Analyser Vulnérabilités potentielles (par degré d'importance : mineure, majeure, bloquante)
- Générer des rapports,
- Compatibilité avec GitLab et GitLab CI (pour de l'intégration continue.. !!)



En conclusion la **vision du software craftsmanship** est une culture digitale qui propose une vision du métier de développeur de qualité et d'apprentissage permanent. Avoir des valeurs autour de ce métier crucial pour les Entreprises, permet de réaliser des logiciels de qualité, opérationnels, bien conçus et économiques. Ces notions, je les mette en pratique dans différents comptes se trouvant sur mon GitHub et entreprise.