1. Donnez une définition de l'agilité?
2. Pourquoi l'Extreme Programming ?
3. Quels avantages y a-t'il à utiliser SCRUM ?
4. Sprint et intégration continue, expliquer le fonctionnement de SCRUM
5. TDD : (Test Driven Development), recherchez des information pour expliquer le principe.

1. L'agilité en programmation est un ensemble de méthodes permettant un travail de développement plus productif, transparent, et compréhensible pour toutes les personnes participant au projet. Il s'agit aussi de faciliter la communication entre les développeurs et les testeurs en centralisant les composants du projet.

2. L'Extreme Programming a été choisi car cela améliore considérablement la qualité et la productivité des développeurs. Le travail en binôme (changeant, on ne travaille jamais avec la même personne participant au projet) permet de déceler les éventuels problèmes dans le code avant même la phase de test, car chaque binôme est composé d'un *driver,* qui aura le clavier entre les mains, et un *partner*, qui pourra aider le *driver* en cas de blocage, ou pour apporter d'autres possibilités.

3. Les avantages de SCRUM sont qu'il permet à chaque personnes de l'équipe de prendre ses responsabilités, et de proposer des itérations plus rapides et moins risquées au client. Les réunions prennent très peu de temps dans la journée du développeur, mais passent en revue tous les problèmes rencontrés par l'équipe ainsi que les éventuelles solutions à appliquer, et aussi les tâches que tout le monde a accompli la veille et compte accomplir dans la journée.

4. En SCRUM les sprints sont des périodes d'une semaine à un mois maximum, durant lesquelles, après une étude du backlog du produit à fournir, l'équipe devra accomplir les tâches du backlog selon l'ordre de priorité, afin de fournir une itération stable et fonctionnelle à la fin de cette période. Il s'agit aussi d'intégrer chaques nouvelles fonctionnalités au fur et à mesure de leur complétion, afin d'éviter un rush pour faire la totalité de l'intégration à la fin du sprint.

5. Le TTD est une méthode de développement. Il s'agit de réfléchir aux tests que chaque fonctionnalité devra passer (tests unitaires) pour être considérée comme fonctionnelle, avant même de commencer à coder. Cette méthode est régie par 3 lois :

Loi n° 1 : Vous devez écrire un test qui échoue avant de pouvoir écrire le code de production correspondant.

Loi n° 2 : Vous devez écrire une seule assertion à la fois, qui fait échouer le test ou qui échoue à la compilation.

Loi n° 3 : Vous devez écrire le minimum de code de production pour que l'assertion du test actuellement en échec soit satisfaite.