**Tests unitaires (méthode TDD)**

* Test une seule fonction technique afin de vérifier que les résultats retournés sont cohérent
* TDD = Test Driven Development = pratique qui consiste à écrire les tests unitaires avant de coder le contenue de la fonction  
    
  Cycle de la TDD :
  + Ecrire le test
  + Lancer celui-ci et vérifier qu’il échoue (classe pas encore codée)
  + Ecrire la classe à tester avec le minimum pour faire marcher le test
  + Lancer le test et vérifier qu’il fonctionne
  + Finir le code complet de la classe
  + Vérifier que le test fonctionne toujours (non régression)

Outils de tests unitaires :

* + JUnit pour JAVA
  + NUnit pour .Net
  + Unit.js pour JS

Outils de lancement automatiques des outils de tests unitaires :

*[Choix à faire en fonction du langage et de la techno]*

* + Team City
  + Jenkins
  + Travis (outils SaaS 🡪 disponible sur le cloud (GitHub ici), pas d’installation nécessaire)

Apports de la TDD :

* + Code de meilleur qualité
  + Diminution drastique des futurs bugs
  + Gain de temps (moins de débogage à faire)

**Tests fonctionnels (méthode BDD / ATDD)**

* But =création de tests fonctionnels avec un langage compris de tous
* Permet de faire une documentation vivante du produit
* Automatisés, ils permettent d’éviter les régressions
* BDD = Behavior Driven Development = écriture des sénarios de tests automatisés  
    
  Outils :
  + Gherkin
* ATTD = Acceptance Test Driven Developement = fait d’écrire les scénarios de test au moment de la définition des Users Stories et d’utiliser ces scénarios comme documentation vivante. (expression du workflow métier)

**Tests end-to-end (e2e)**

* Permet de simuler l’utilisation d’une application par un utilisateur clé en suivant un parcours type.
* Permet de détecter le moindre bug de comportement, mais on peut limiter les tests aux parties critiques pour gagner du temps (cependant ces tests sont généralement réalisés par le Product Owner ou par des testeurs)
* Outil : Selenium

**Tests de performances**

* Permet de s’assurer que l’application réponde dans un temps défini
* Outils : gatling
* Mise en place rapide, rentabilité assurée

**Tests de charges**

* Permet de tester la capacité d’une application à ne pas cracher dans différentes situations
* Test classique : on augmente considérablement le nombre d’utilisateurs et on vérifie que l’application réponde correctement.
* Test de stress : on observe le comportement de l’application face à une charge extrême.
* Tests de capacité : test de stress dans lequel on mesure le nombre d’utilisateur critique pour chaque fonctionnalité.
* Test d’endurance : tester le comportement de l’application lors d’un test de charge sur la durée
* Type de test en général utile et vite rentabilisé
* Outil : gatling**Tests de qualimétrie**
* Outil : SonarQube