TEST DRIVEN DEVELOPMENT

Roozbeh SOLTANI

Roozbehsltn@gmail.com

On parle de ...

- Qu'est ce que c'est la *Test-Driven Development* (TDD)
- Cycles de TDD
- **Les Trois Lois de TDD**
- Comment utiliser le Test driven development ?
- Exemple de Test driven development, TDD
- **Quel que point à propos de TDD**
- Processus cyclique de développement
- Des avantages
- **Comparaison avec le cas traditionnel**
- **TDD en java**
- **TDD en php**
- **TDD en C#**
- **Conclusion**

Test-Driven Development (TDD)

- La TDD (Test Driver Development) est une technique de développement logiciel qui consiste à écrire les tests unitaires d'une fonction avant d'écrire le contenu de cette fonction. C'est une pratique très connue dans le domaine de l'intégration continue.

- Le but de cette pratique connue en XP, Scrum XP ou en devops est d'indiquer à quoi s'exposera le code avant même d'être écrit. Cette pratique change radicalement les concepts classiques des développeurs et peut s'avérer perturbante.

Test-Driven Development (TDD)

- Bien que le TDD renforce la qualité des produits et limite les futures regressions, la notion de « test » n'est pas sa principale fonction. Le TDD permet principalement d'amener les développeurs à trouver une solution à une demande posée par le test.
- En effet, en réalisant le test en amont, les développeurs seront guidés pour aller vers la solution attendue. Terminé les pertes de temps pour réaliser des fonctionnalités avec le risque fort d'arriver à un résultat non satisfaisant.

Cycles de TDD

La TDD propose un cycle de travail aux développeurs pour obtenir une qualité optimale de la mise en place des tests unitaires :

- Ecrire le test unitaire
- 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)

Les Trois Lois de TDD

1. Loi nº 1 : Vous devez écrire un test qui échoue avant de pouvoir écrire le code de production correspondant.
2. Loi nº 2 : Vous devez écrire une seule assertion à la fois, qui fait échouer le test ou qui échoue à la compilation.
3. Loi nº 3 : Vous devez écrire le minimum de code de production pour que l'assertion du test actuellement en échec soit satisfaite.

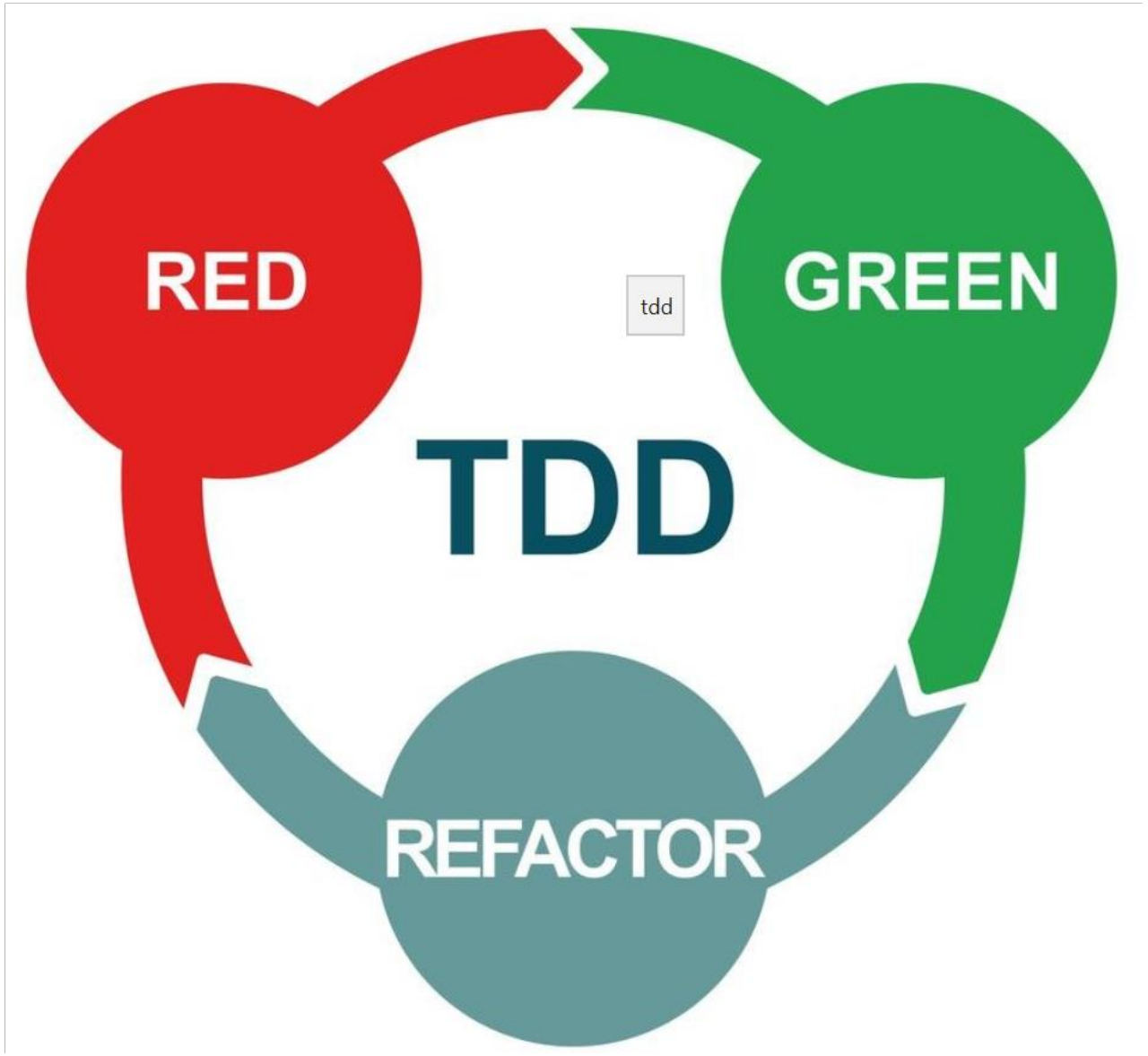
Comment utiliser le Test driven development ?

Le concept du TDD préconise un processus de développement en cinq phases distinctes :

- rédiger un test unitaire ne décrivant qu'une seule partie de la problématique posée ;
- confirmer l'échec du test ;
- écrire cette fois-ci un morceau de code suffisant pour qu'il réussisse le test ;
- confirmer la réussite du test ;
- améliorer le code et contrôler l'absence de régression.

Exemple de Test driven development, TDD

- Plus concrètement, le Test driven development se déroule par cycle de trois phases : Rouge (Red), Green (Vert) et Remaniement (Refactor). Selon cette méthode, le développeur du logiciel doit commencer par écrire un test unitaire. Par essence, ce dernier va obligatoirement échouer puisque le code testé n'a pas encore été créé. Le test unitaire est donc symbolisé par la couleur rouge. Dans un second temps, le programmateur écrit suffisamment de code pour que le test unitaire réussisse et passe ainsi au vert dans le système. Enfin, l'étape du remaniement, elle, nécessite que le développeur remanie son code en contrôlant que tous les tests qu'il réalise restent bien en vert.


RED
GREEN
tdd
TDD
REFACTOR

Quel que point

- On fait ainsi la distinction entre l'assertion à l'origine de l'échec du test et le test qui échoue lors de son exécution.
- Le fait de mettre bout à bout les 3 lois de TDD en une seule itération constitue ce qui est appelé un nano-cycle de TDD.
- À noter que ces 3 lois ne couvrent que les conditions à respecter en TDD pour arriver à un test qui réussit, en exprimant le minimalisme attendu du test en échec.

Processus cyclique de développement

Le processus préconisé par TDD comporte cinq étapes :

- écrire un seul test qui décrit une partie du problème à résoudre ;
- vérifier que le test échoue, autrement dit qu'il est valide, c'est-à-dire que le code se rapportant à ce test n'existe pas ;
- écrire juste assez de code pour que le test réussisse ;
- vérifier que le test passe, ainsi que les autres tests existants ;
- puis remanier le code, c'est-à-dire l'améliorer sans en altérer le comportement.

Ce processus est répété en plusieurs cycles, jusqu'à résoudre le problème d'origine dans son intégralité. Ces cycles itératifs de développement sont appelés des micro-cycles de TDD.

Intérêt

- Les tests tels qu'ils sont mis à profit dans TDD permettent d'explorer et de préciser le besoin
- spécifier le comportement souhaité du logiciel en fonction de son utilisation, avant chaque étape de codage
- On obtient donc un logiciel mieux conçu, mieux testé et plus fiable, autrement dit de meilleure qualité
- Comme chaque test correspond à un changement minimal de code, un test permet de faire un lien évident entre une régression et sa source s'il échoue

Intérêt

- Un guide pour les développeur
- Une pratique pour diminuer considérablement les bugs
- Un gain de temps dans un avenir proche pour faire du projet et non plus du débogage
- Une pratique de développement très appréciée sur un CV

Comparaison avec le cas traditionnel

- Quand les tests sont écrits après codage, comme c'est le cas traditionnellement, les choix d'implémentation contraignent l'écriture des tests : les tests sont écrits en fonction du code, et si certaines parties du code ne sont pas testables, elles ne seront pas testées. Au contraire, en testant avant de coder, on utilise le code avant son implémentation, de sorte que les contraintes définies par les tests sont imposées à l'implémentation : le code est écrit en fonction des tests
- Le fait d'écrire les tests avant le code en TDD est donc à l'origine d'implémentations testables, c'est-à-dire facilement testables et testables à 100%. Or la testabilité du code favorise une meilleure conception par un couplage lâche et une cohésion forte, ce qui évite des erreurs communes de conception.

Outils de tests unitaires

- Chaque langage possède son outil de test unitaire facilement exploitable et tous basés sur les mêmes principes.
  - Php Unit (PHP)
  - JUNit (Java)
  - NUnit (.Net)
  - Unit.js (Js)
  - CppUnit (C++)

Test driven development en Java

- De nombreux développeurs professionnels de logiciels utilisent le langage Java pour concevoir leurs programmes. Son principal atout ? Sa portabilité entre les différents systèmes d'exploitation : Windows, MacOS, Unix... Il sert également de base à la grande majorité des applications en réseau.

Test driven development en C#

- Distribué par Microsoft depuis 2002, C# est un langage de programmation proche de Java. Il est surtout employé pour des applications et des services Web, mais aussi des widgets et quelques applications de bureau. C# est destiné à être développé sur la plateforme Microsoft .NET.

Test driven development en PHP

- Le langage de programmation libre PHP: Hypertext Preprocessor, plus connu sous son acronyme PHP, est pour sa part utilisé pour développer des pages Web dynamiques via un serveur http. C'est un langage orienté objet qui est à l'origine d'un grand nombre de sites Internet.

Conclusion

- TDD permet d'éviter des modifications de code sans lien avec le but recherché, car on se focalise étape par étape sur la satisfaction d'un besoin, en conservant le cap du problème d'ensemble à résoudre.
- TDD permet d'éviter les accidents de parcours, où des tests échouent sans qu'on puisse identifier le changement qui en est à l'origine, ce qui aurait pour effet d'allonger la durée d'un cycle de développement.
- TDD permet de maîtriser le coût des évolutions logicielles au fil du temps, grâce à une conception du code perméable au changement.
- TDD permet de s'approprier plus facilement n'importe quelle partie du code en vue de le faire évoluer, car chaque test ajouté dans la construction du logiciel explique et documente le comportement du logiciel en restituant l'intention des auteurs.
- TDD permet de livrer une nouvelle version d'un logiciel avec un haut niveau de confiance dans la qualité des livrables, confiance justifiée par la couverture et la pertinence des tests à sa construction.