Les bases du TDD

Lionel DURAND

Avril 2023

lioneldurand@gmail.com
https://www.linkedin.com/in/lionel-durand-649b72193/

Ice breaker

- Vos attentes ?
- Vos questions p/r au TDD ?
- Comment vous pratiquez les tests aujourd'hui ?
- Vos douleurs p/r aux tests aujourd'hui ?

Plan

- J1 : Les bases du TDD
  - Kata Password Verifier
  - Mock & stub, Kata Greeting service
- J2 : TDD sur du code existant
  - Kata Trip service
  - Kata Tennis
- J3 : TDD avec dépendances
  - Kata Roulette
- J3 Bonus
  - TDD sur votre code / TDD legacy advanced

TDD : VRAI / FAUX ?

TDD est une technique de test qui consiste à couvrir de TU un programme

TDD : VRAI / FAUX ?

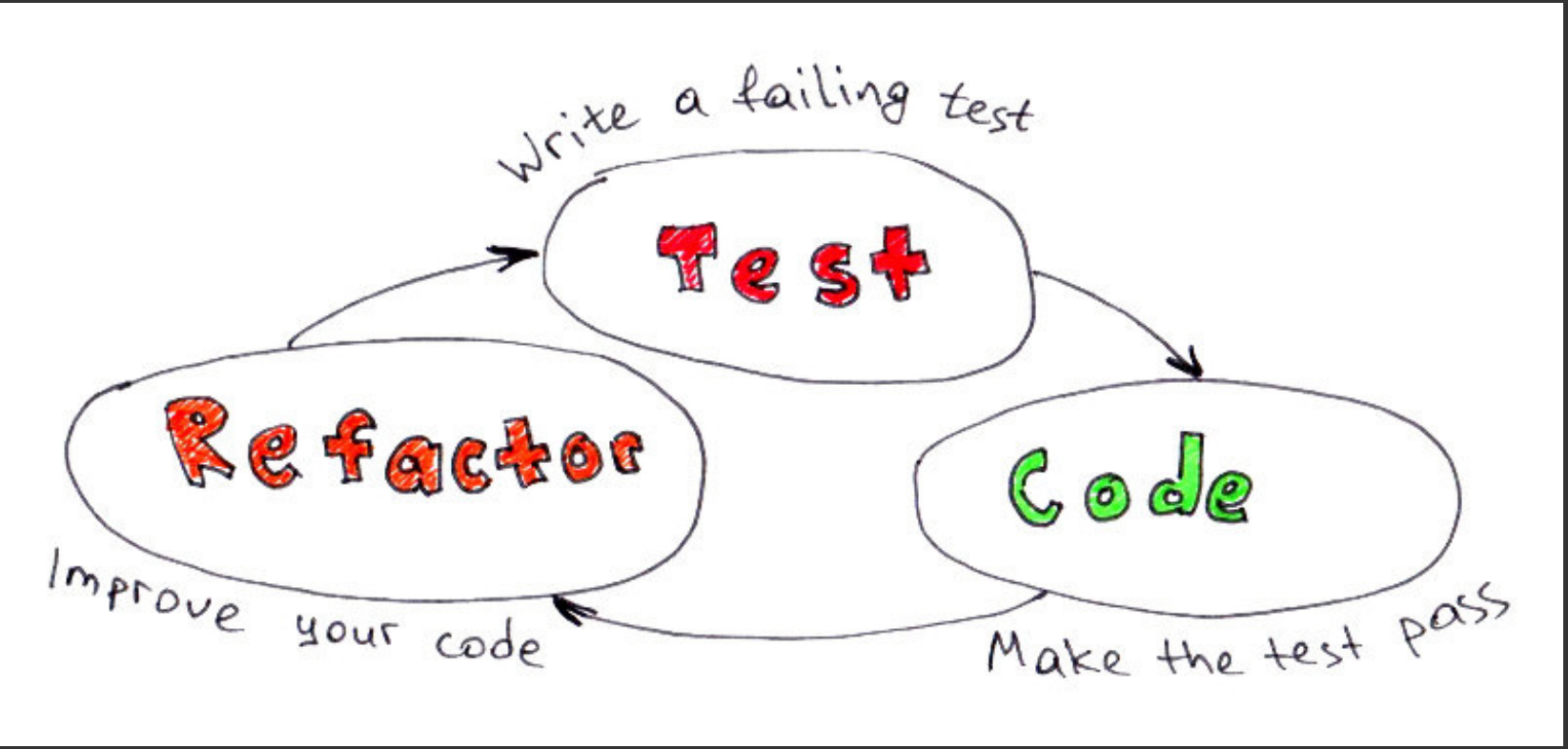
~~*TDD est une technique de test qui consiste à couvrir de TU un programme*~~

TDD est une technique de développement qui utilise les tests d'abord comme un moyen pour spécifier

TDD = Test Driven Development

- TDD met l'accent non sur les tests, mais sur l'expression l'intention
- C'est une pratique qui vise à aider le développeur à réfle avant d'écrire son code
- La couverture par les tests du code ainsi réalisé est un bénéfice collatéral de la méthode, et non son but

TDD : le cycle red/green/refactor



TDD : Comment ?

- identifier les tests à écrire
- pas de code sans test préalable
- le test doit d'abord être rouge
- écrire juste le code nécessaire pour faire passer le test... plus
- puis refactorer, en apportant le même soin au code des qu'au code fonctionnel
- faire des petits pas, revenir en arrière (au vert) si besoin

TDD : Pourquoi ?

- Feedback temps réel : plus de "tunnel" de dév, debuggi simplifié
- Refactoring facilité/permanent, faisable sur du legacy
- Documentation par les tests
- Design testable => design modulaire

Voir les 3 lois du TDD ou la Conference Advanced TDD, par Uncle Bob

TDD : Coûts / Benefices

Table 1. Systematic literature reviews on test-driven development (TDD).

Study	Overall conclusion for quality with TDD	Overall conclusion for productivity with TDD	Inconsistent results in the study categories
Bissi et al.[3]	Improvement	Inconclusive	Productivity: Academic vs. industrial setting
Munir et al.[4]	Improvement or no difference	Degradation or no difference	Quality: • Low vs. high rigor • Low vs. high relevance Productivity: • Low vs. high rigor • Low vs. high relevance
Rafique and Mišić[5]	Improvement	Inconclusive	Quality: Waterfall vs. iterative test-last Productivity: • Waterfall vs. iterative test-last • Academic vs. industrial
Turhan et al.[6] and Shull et al.[1]	Improvement	Inconclusive	Quality: • Among controlled experiments • Among studies with high rigor Productivity: • Among pilot studies • Controlled experiments vs. industrial case studies • Among studies with high rigor
Kollanus[7]	Improvement	Degradation	Quality: • Among academic studies • Among semi-industrial studies
Siniaalto[8]	Improvement	Inconclusive	Productivity: • Among academic studies • Among semi-industrial studies

Extrait de IEEE Software - Juillet/Aout 2018 What Do We (Really) Know about Test-Driven Development?

TDD : Coûts / Benefices

- Amélioration de la qualite interne / externe
- Parfois impact négatif sur la productivité
- Des contradictions entre les differentes études, influenc contexte
- Niveau d'adherence au "protocole" TDD non précisé pa études

=> Developper son propre jugement sur ce qu'on peut attendre du TDD dans un contexte donnée. Faire des essais, évaluer le résultat et ajuster.

Exemple : Prime factors

Décomposer en facteurs premier un nombre n

Quels tests ?

- 1 => []
- 2 => [2]
- 3 => [3]
- 4 => [2*2]
- 15 => [3*5] ...

https://github.com/duduyo/kata-prime-factors-java

Prime factors : bilan

- choisir les tests, leur ordre
- l'intention avant l'implémentation = construire les composants depuis l'extérieur
- petits pas dans les tests / petits tests
- petits pas lors du refacto : ne pas casser
- attendre pour refactorer ("just in time")
- maléabilité du code
- outillage (IDE)

Mise en pratique

Kata "Password validator"

Pour commencer :

https://github.com/duduyo/kata-bootstrap-java

Validation de mot de passe

Créer un classe qui valide des mots de passe. Pour être valide, un mot de passe doit :

- être composée de plus de 8 charactères
- contenir au moins une lettre
- contenir au moins un chiffre
- contenir au moins un character special

Considérer que le mot de passe à valider n'est jamais null.

Rappel des règles

- identifier les tests à écrire
- pas de code sans test préalable
- le test doit d'abord être rouge
- écrire juste le code nécessaire pour faire passer le test... plus
- refactorer, en apportant le même soin au code des tests qu'au code fonctionnel
- faire des petits pas, revenir en arrière (au vert) si besoin

Fin de sprint 1 : Demo

Verifier les mots de passe suivants

- 1#Abcdefgh
- 1#Abcdef
- /Aé123456
- @Ag123456

Fin de sprint : Retrospective

- Quelles difficultés avez vous rencontrées?
- Comment vous les avez addressées? Comment on peut remedier?

Evolutions

- Le mot de passe doit contenir au moins 10 charactères
  - Combien de tests est il nécessaire de modifier?
  - Est-ce qu'on peut améliorer le design pour intègr facilement cette évolution?
- Voir un message qui indique les règles non respectées

Evolutions : acceptance tests

- 1#Abcdefgh => OK
- 1#Abcdefg => KO
- x#Abcdefgh => KO
- 1xAbcdefgh => KO
- 1#12345678 => KO

Parallel Change

Technique pour l'ajout des changements non-retrocompatibles à une interface

- ajouter un nouvelle méthode metant en oueuvre la nou interface
- migrer progressivement les clients
- supprimer l'ancienne méthode

https://martinfowler.com/bliki/ParallelChange.html

Exemples de solution

https://github.com/duduyo/kata-password-java

Points clé

- cycle test/code/refactor
- avancer pas à pas
- écouter les tests : si le code est difficile à tester, il y a probablement un problème sur le design
- difficulté à tout tester au plus haut niveau

Stubs and Mocks

Kata "Greeting service"

https://github.com/duduyo/kata-greeting-service-java

Stubs and mocks : points clés

- font partie des [test doubles](http://xunitpatterns.com/Double.html)
- utiles / nécessaires pour tester un composant qui depend d'autres composants
- remplacent le composant réel pour les besoins des tests
- **stubs** : fournissent des données nécessaires au test
- **mocks** : permettent de vérifier que le composant sous-test effectue certains appels
- d'autres tests doubles : fakes, dummies, ...

L'IDE un outils précieux

- Pour lancer les tests
- Pour lancer les tests en continu (voir aussi Infinitest)
- Pour analyser la couverture de test (coverage)
- Pour automatiser le refactoring ou la génération de cod
- ~~Pour débugguer~~

IDE : refacto et génération de code

L'IDE sait faire beaucoup de choses

- générer du code / le modifier
- travailler sur les conditions
- détecter les duplications, etc...

Intérêt

- Le refacto devient plus fiables / rapide
- Encore plus via raccourcis clavier
- Il suffit de connaitre une dizaine d'actions (Voir ici)

IDE : Raccourcis

- Inellij, menu Help/Keymap Reference
- Find actions : Ctrl+Maj+A / Command+Maj+A
- Plugin KeyPromoter X