TDD

Test Driven Development

Plan

- Présentation rapide
- Démonstration
- Premier exercice
- Points complémentaires
- Deuxième exercice
- Conclusion

TDD

- Le sujet principal n’est pas le test
  - Il ne faut pas voir le TDD comme une manière de faire des tests

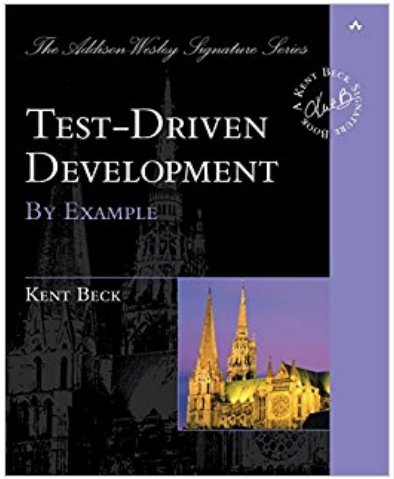
- C’est un process de développement guidé par les exemples
  - Le principe est de développer en partant de cas concrets
  - Les exemples successifs mettent en évidence les manquent dans l’implémentation
- L’objectif est de maintenir un rythme de développement
  - Les évolutions du logicel doivent rester gérable dans le temps

Kent Beck

- Extreme programming
- Manifeste agile
- Test-driven development
- JUnit

https://fr.wikipedia.org/wiki/Kent_Beck

Préface



- N’écrivez pas une ligne de code tant que vous n’avez pas d’abord un test automatique qui échoue
- Eliminez la duplication

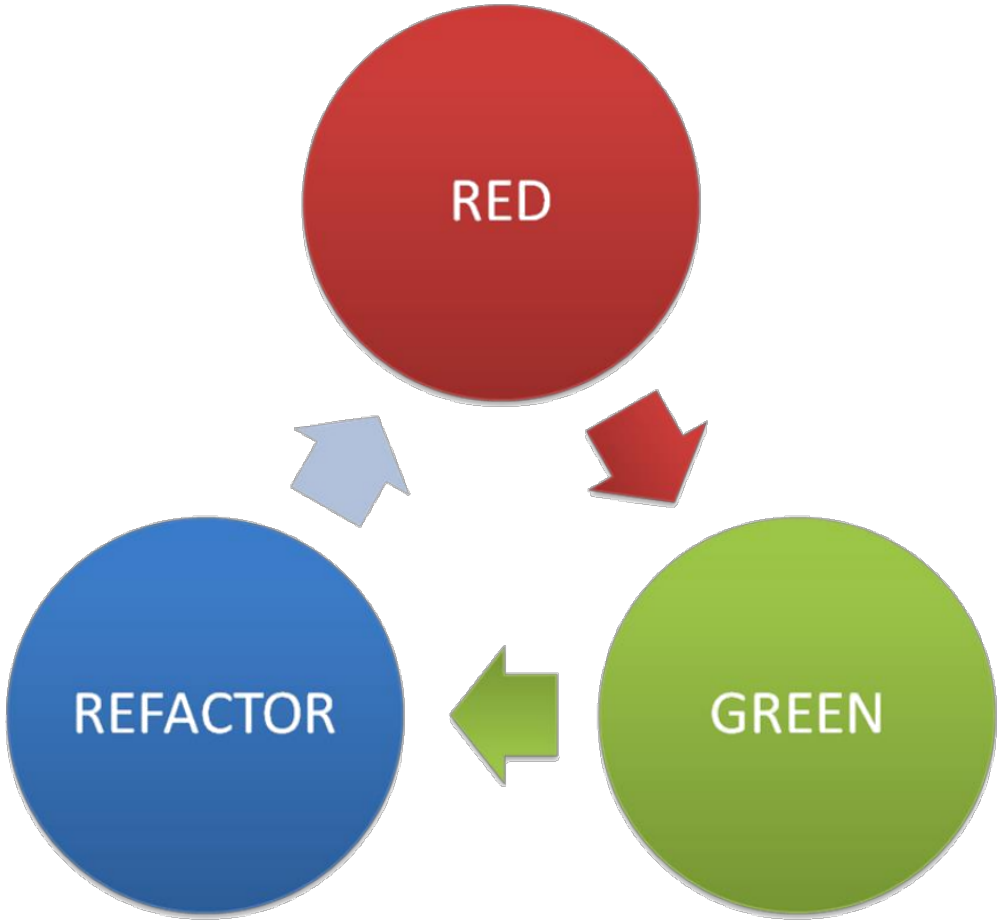
NOTE

Kent Beck est l’auteur de "TDD by example"

Ces 2 phrases sont issues de la préface du livre

Il est ajouté que "ces 2 règles simples entrainent des comportements individuels complexes"

TDD mantra



TDD mantra

- Rouge: Ecrire un test qui échoue et qui prouve qu'il faut modifier le code
- Vert: Faire passer le test avec une implémentation rapide
- Bleu: Remanier le code pour simplifier les prochaines évolutions

Un exemple

- Une calculatrice gérant l’addition de plusieurs entiers

Concepts clés

- Ecrire un test et s'assurer qu'il échoue
  - Les valeurs en « dur » orientent le test suivant
- Ecrire « du » code pour satisfaire le test
  - Passer rapidement au vert
  - Implémentation rapide
- Ecrire « le » code que l'on va garder
  - Renommage, restructuration
  - Changement d'implémentation

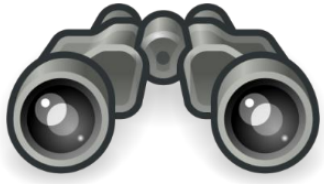
Questions

Premier exercice - FizzBuzz

- Implémenter la fonction FizzBuzz qui prend en paramètre un nombre entre 1 et 100
- Par défaut, retourner le nombre sous forme de chaîne de caratères
- Si le nombre est un multiple de 3, retourner Fizz
- Si le nombre est un multiple de 5, retourner Buzz
- Pour les nombres multiple de 3 et de 5, retourner FizzBuzz

NOTE	Pour apprendre le TDD, il faut le pratiquer

Rétrospective



Points d’attention

- Se focaliser sur le comportement/besoin et non la manière
- Penser utilisation avant implémentation
- Ne pas modifier le test et le code en même temps
- Prendre autant soin des tests que du code

NOTE Points importants pour la mise en place du TDD

Bonnes pratiques de test

- Indépendance
- Rapidité d’exécution
- Reproductibilité
- Lisibilité
- Tester une seule chose à la fois
- Le nom du test indique l’objectif

NOTE	On va écrire beaucoup de test. Il est important de les écrire correctement pour qu’ils restent maintenable et utiles.

Outillage

- Framework de tests: Junit, TestNG
- Outils de build: Maven, Ant
- IDE: Eclipse, IntelliJ, NetBean
- Intégration continue: Jenkins, Travis CI, GitLab CI
- Couverture de code: Cobertura, Emma
- Analyse de code: Sonar, Checkstyle, PMD, ...
- Plugins: MoreUnit, Inifinitest

NOTE

La pratique du TDD vise à garder un code propre Les frameworks de tests permettent d'exécuter les tests et produire des rapports. Les outils de build peuvent intégrer l'exécution des tests (mvn test) L'IDE permet l'execution des tests et fourni des outils de refactoring L'intégration continue permet de savoir en permanence dans quel état se trouve le projet La couverture de code n'a que peut d'intérêt en TDD mais il est toujours intéressant de l'avoir L'analyse de code permet d'identifier des mauvaises pratiques, voir des bugs Des outils complémentaire aident à la productivité et au feedback rapide

Spécification / Documentation

- Spécification exécutable
  - Code lisible
  - Indépendant de l'implémentation
- Documentation à jour
  - Exemple d'utilisation du code
  - Spécification du comportement

NOTE

Il faut voir les tests comme une spécification/documentation exécutable Il faut donc lui apporté de l'attention pour qu'ils soient lisibles Il faut tester les comportements et non l'implémentation pour permettre le refactoring

Concepts agiles

- KISS (Keep It simple, stupid)
  - On commence par une implémentation triviale
  - On restructure pour simplifier
- YAGNI (You Ain’t Gonna Need It)
  - On ne développe que ce qui est nécessaire pour faire passer un test
  - On écrit un test que pour décrire un cas utilisateur

NOTE Le TDD s’inscrit pleinement dans un vision agile du développement.

Feedback

- Baby steps
  - Approche itérative très courte
  - Construction organique
- Feedback
  - Retour immédiat
  - Rythme de développement

NOTE

Le développement se fait de manière organique en opposition avec une vision "sur plan" On doit développer en état capable de modifier à tout moment et de garder cette capacité

Le feedback rapide permet de savoir où on en est Il permet d’avancer continuellement avec des objectifs courts qui maintiennent le rythme

Mesure de la couverture

- La couverture est assurée par construction
- On ne s’en préoccupe pas spécialement

NOTE

Tout code étant écrit de manière minimal suite à un test qui échoue La couverture doit donc être proche de 100% mais surtout chaque test montre un cas particulier à gérer La mesure de la couverture est très relative car elle montre que l’on passe dans le code plus que le code est testé

Qualité de code

- Refactoring
  - Modification de l’implémentation sans changer le comportement
  - Elimination de la duplication
  - Amélioration de l’implémentation

NOTE

La qualité de code est un point essentiel pour maintenir le code dans un état gérable. Elle se dégrade fatalement au fil des ajouts Le TDD permet de conserver cette qualité de code en permettant un refactoring sécurisé.

BDD: Behavior Driven Development

- Continuité du TDD
- Encore plus orienté vers le métier
- Rédaction en collaboration avec le métier
- Syntaxe Gerkhin: Given / When / Then

```
Scénario: Compléter
toute ma todo liste
  Etant donné que j
'ai 2 tâches dans ma
todo liste
  Lorsque je compl
ète toutes mes t
âches
  Alors ma todo
liste est vide
```

Second exercice

- Bowling Game
- Game of life
- How much water ?
- Tennis score

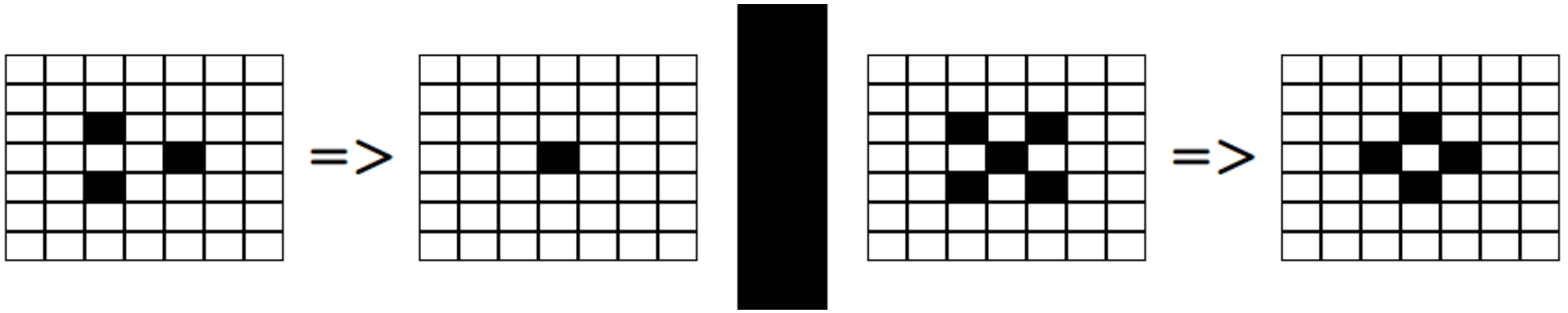
Score de bowling

- La grille de bowling est constituée de 10 cadres.
- Pour chaque cadre, le joueur à deux lancés pour faire tomber les 10 quilles.
- Le score du cadre est le nombre de quilles tombées plus un bonus en cas de spare ou de strike.
- Il y a spare lorsque qu'un joueur fait tomber toutes les quilles en deux coups. Le bonus est le nombre de quilles tombées au coup suivant
- Il y a strike lorsque toutes les quilles tombent au premier essai. Le bonus est le score des deux coups suivants.

	1	5	3	1	6	/	2	1	0	5	X	-	3	4								
Joueur A	**6**		**10**		**22**		**25**		**30**		**47**		**54**									

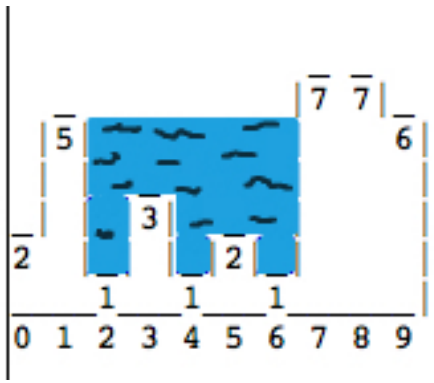
Le jeu de la vie: John conway

- **Pour un emplacement ‘peuplé':**
  - Une cellule avec un ou aucun voisin meurt de solitude.
  - Une cellule avec quatre voisins ou plus meurt de surpopulation.
  - Une cellule avec deux ou trois voisins survit.
- **Pour un emplacement ‘vide’ ou ‘non peuplé’**
  - Une cellule avec trois voisins devient peuplée.



How much water ?

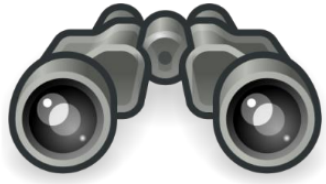
- Etant donnée une liste d’entiers représentant les hauteurs de colonnes
- On cherche la quantité d’eau qui resterait prisonnière des cuvettes formées par les colonnes



Tennis score

- Afficher le score d’un match de tennis
- On donne une suite indiquant qui a marqué chaque point et on retourne le score
- Exemples
  - AAAB ⇒ 40 - 15
  - AABB ⇒ 30A
  - BBBB ⇒ Jeu B

Rétrospective



Les points difficiles

- Les méthodes privées
- Les contributeurs
- Rester indépendant de l'implémentation
- Tester sur du code existant
- Conception émergente

Bénéfices

- Composants prévus pour être testés
- Composants prévus pour être réutilisés
- Capacité à faire évoluer/modifier le code
- On sait ce qui marche ou pas
- Projet auto validé
- Rapidité d’analyse des défauts

Bénéfices

- Le tests ne sont plus une option "lorsqu’il reste du temps"
- On ne perd pas du temps à écrire les tests, on gagne du temps pour écrire le code

Références

- ***Extreme programming explained: embrace change***

  Kent Beck, Addison-Wesley, 1999

- ***Test-Driven Development: By Example***

  Kent Beck. Addison-Wesley, 2002

- ***Test-Driven Development: A Practical Guide***

  David Astels. Prentice Hall, 2003

- ***Growing Object-Oriented Software, Guided by Tests***

  Steve Freeman, Nat Pryce, 2009

Sites

- Cyber dojo: http://cyber-dojo.org/
- CodingDojo: http://codingdojo.org/kata/
- Yosethegame: http://yosethegame.com/
- Coding Game: https://www.codingame.com/start

Resources

- Vidéos
  - TDD : pour que votre code soit testable et testé! - Xavier Nopre, 2019
  - Xavier Screen Cast #3, Premier test unitaire et intro TDD, présentation du TDD à partir de 6mn

    - Test-Driven Development (TDD) in Python #1 - The 3 Steps of TDD - Jason Gorman

- Autres
    - TDD - Jason Gorman, 2016

Meetups

- https://www.coderetreat.org
- https://www.meetup.com/fr-FR/Craft-your-Skills/
- https://www.meetup.com/fr-FR/CARA-Dojo/
- https://www.meetup.com/fr-FR/Software-Craftsmanship-Lyon

Pour apprendre le TDD, il faut le pratiquer