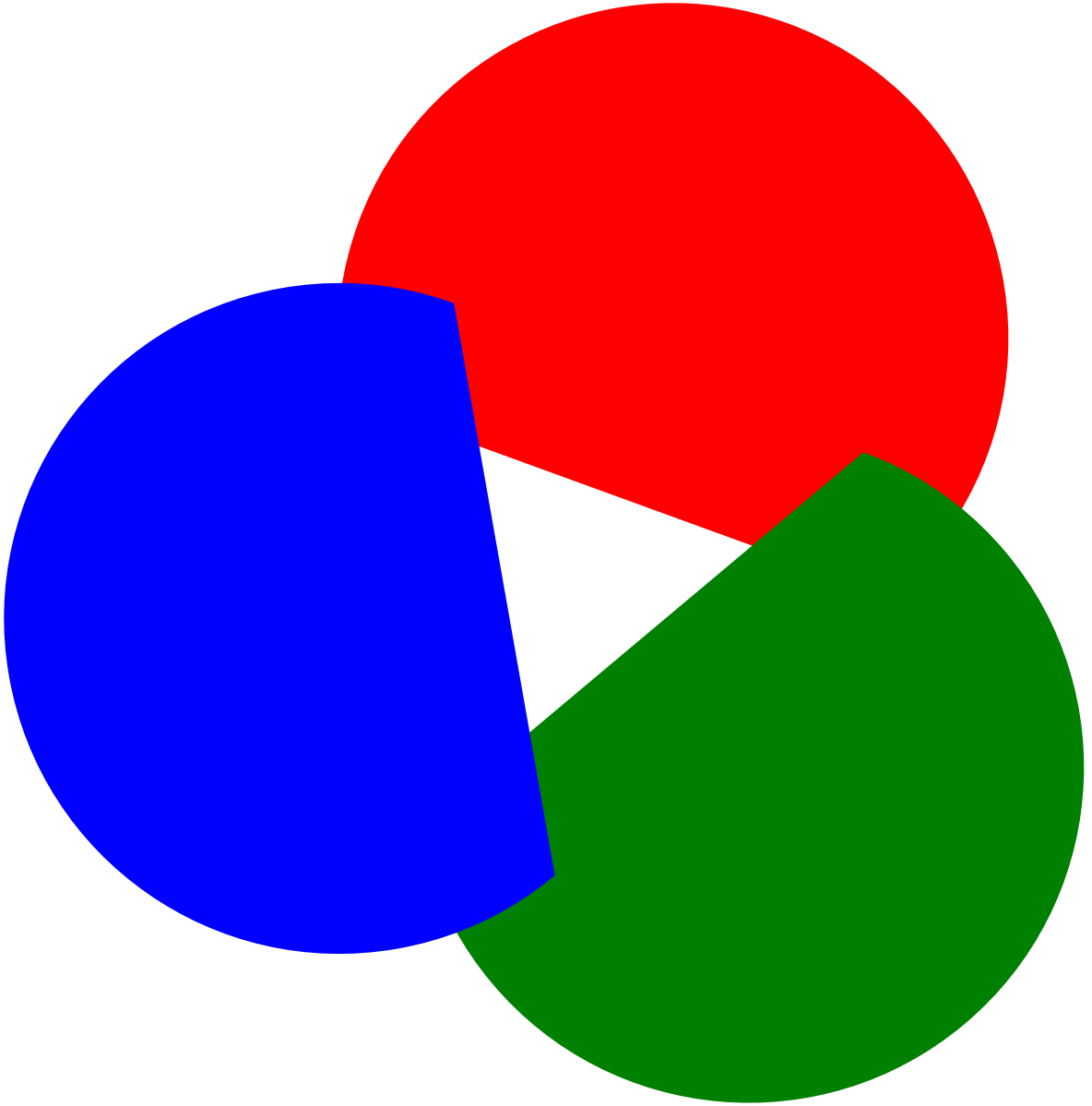
Test Driven Development



Plan

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

TDD

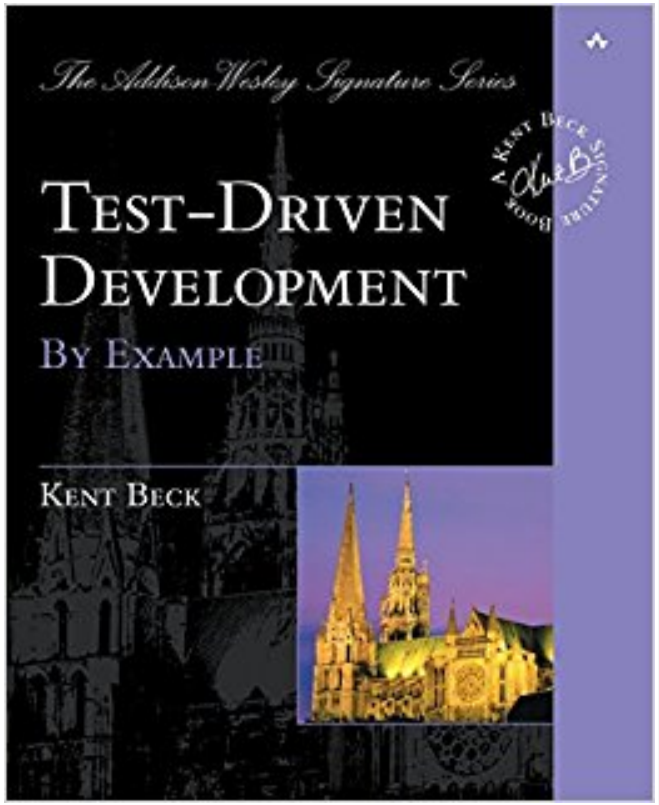
- Le sujet principal n’est pas le test
  - Il ne faut pas voir le TDD comme une manière de faire des tests
- C’est un process de développement guidé par les exemples
  - Le principe est de développer en partant de cas concrets
  - Les exemples successifs mettent en évidence les manquent dans l’implémentation
- 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

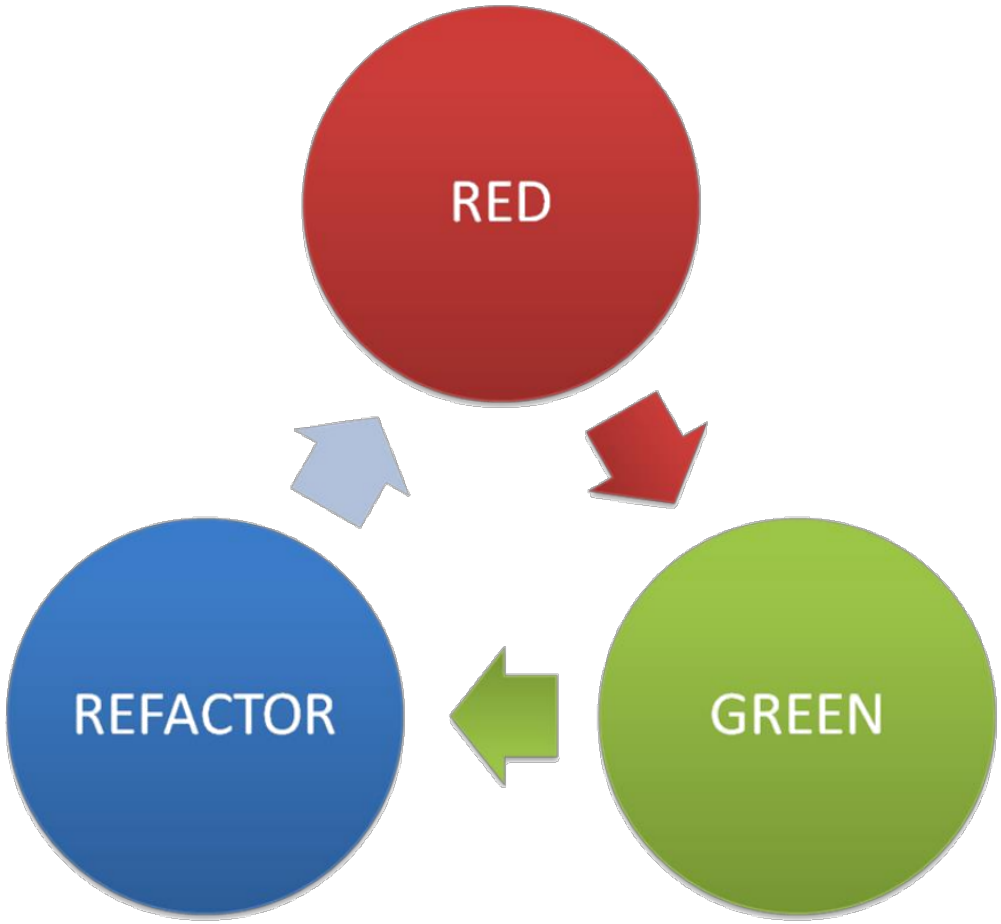
https://fr.wikipedia.org/wiki/Kent_Beck

Préface



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

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

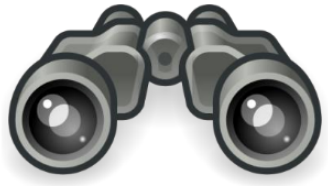
- Ecrire un test et s'assurer qu'il échoue
  - Les valeurs en « dur » orientent le test suivant
- Ecrire « du » code pour satisfaire le test
  - Passer rapidement au vert
  - Implémentation rapide
- Ecrire « le » code que l'on va garder
  - Renommage, restructuration
  - Changement d'implémentation

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

Rétrospective



Points d’attention

- Se focaliser sur le comportement/besoin et non la manière
  - Décrire les cas métier sans penser à la manière de les coder
- Penser utilisation avant implémentation
  - Réfléchir à comment on veut utiliser l’API
- Ne pas modifier le test et le code en même temps
  - Faire des petits pas
  - Rester dans un état stable en permanence
- Prendre autant soin des tests que du code
  - Les tests sont aussi importants que le code

Bonnes pratiques de test

- **Indépendance**: Les tests peuvent être exécuté tout ou partie et sans ordre spécifique.
- **Rapidité d'exécution**: Pour être joués en permanence, ils doivent être **très** rapides.
- **Reproductibilité**: Le résultat doit être le même quelque soit les conditions d'exécution.

Bonnes pratiques de test

- **Lisibilité**: On doit comprendre ce qui se passe uniquement en lisant le test.
- **Tester une seule chose à la fois**: un test en échec ne doit avoir qu'une raison d'échouer.
- **Le nom du test indique l'objectif**: On sait ce que fait un test simplement en lisant son nom.

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

Spécification / Documentation

Les tests sont avant tout une suite d'exemples du fonctionnement de l'application. Ils doivent permettrent de comprendre comment le logiciel marche sans aller voir le code.

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

Concepts agiles

Le TDD permet de mettre oeuvre une certaines philosophie portée par l'agilité.

- 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

Feedback

Le TDD permet de développer progressivement une solution, tout en sachant à tout moment où on en est.

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

Mesure de la couverture

La couverture de code permet d'identifier les lignes de codes exécutées et de mettre en évidence des lacunes. Elle n'assure pas la pertinence des tests. L'objectif doit être de se rassurer sur le bon fonctionnement avant de respecter une métrique.

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

Qualité de code

Le TDD rend possible la modification du code de manière sécurisée. C'est le prérequis pour limiter la dégradation du code au fil des évolutions et pour conserver un logiciel maintenable dans le temps.

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

BDD: Behavior Driven Development

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

```
Scénario: Compléter toute ma todo liste
  Etant donné que j'ai 2 tâches dans ma todo liste
  Lorsque je complète toutes mes tâches
  Alors ma todo liste est vide
```

Second exercice

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

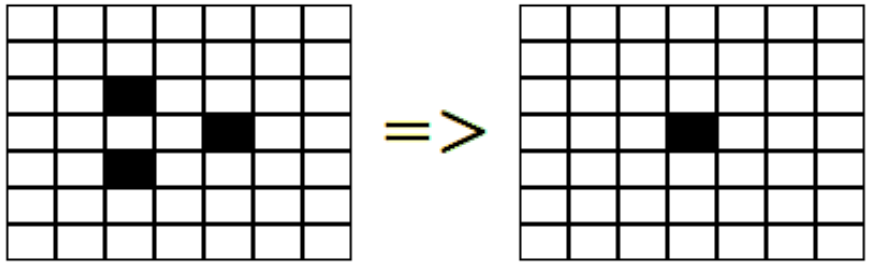
Score de bowling

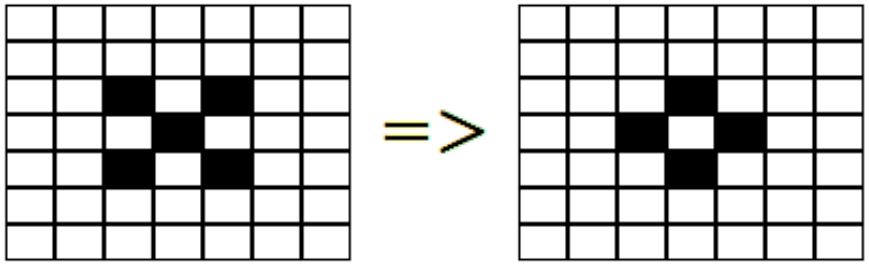
- La grille de bowling est constituée de 10 cadres.
- Pour chaque cadre, le joueur à deux lancés pour faire tomber les 10 quilles.
- Le score du cadre est le nombre de quilles tombées plus un bonus en cas de spare ou de strike.
- Il y a spare lorsque qu'un joueur fait tomber toutes les quilles en deux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

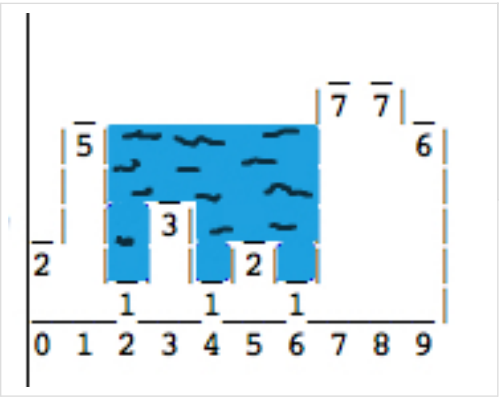
- **Pour un emplacement ‘peuplé':**
  - Une cellule avec un ou aucun voisin meurt de solitude.
  - Une cellule avec quatre voisins ou plus meurt de surpopulation.
  - Une cellule avec deux ou trois voisins survit.
- **Pour un emplacement ‘vide’ ou ‘non peuplé’**
  - Une cellule avec trois voisins devient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



- Exemple :
  - Valeurs: 2, 5, 1, 3, 1, 2, 1, 7, 7, 6
  - Résultat: 17

Tennis score

- Afficher le score d’un jeu au tennis à partir d’une liste indiquant qui a marqué chaque point
- Les points valent 15, 30, 40
- Au delà de 40,
  - le premier avec 2 points d’écart gagne le jeu: "Game [A ou B]"
  - s’il n’y a pas 2 points d’écart, on affiche "Advantage [A ou B]"
  - sil y a égalité, on affiche "Deuce"

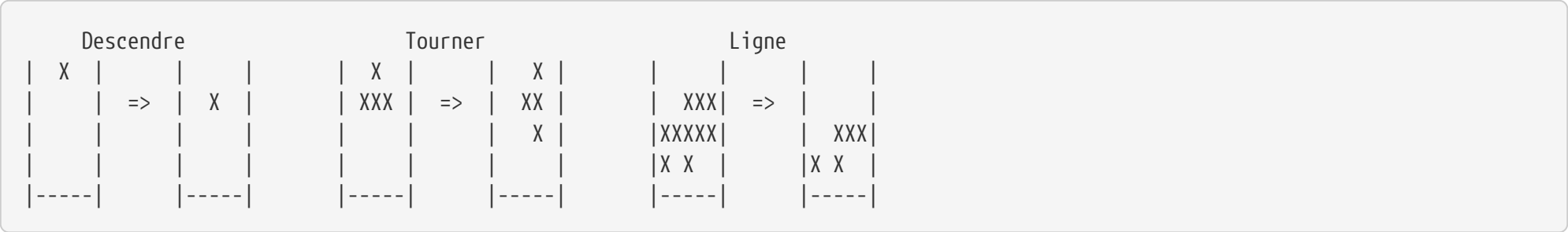
Exemples

AAAB ⇒ 40 - 15	BBBB ⇒ Game B	ABABABA ⇒ Advantage A

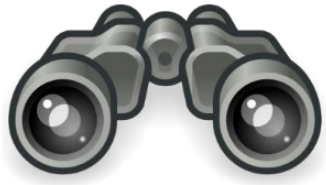
Tetris

- Afficher l'état
- Déplacer une pièce (bas, droite, gauche)
- Détecter les mouvements impossibles
- Faire tourner une pièce
- Supprimer les lignes complètes

Exemples



Rétrospective



Les points difficiles

Certains points demandent un peu d'expérience avant d'être à l'aise pour les adresser.

- 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
- TDD: https://codemanship.wordpress.com/category/tdd/
  Jason Gorman

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
- Unit Testing Is The BARE MINIMUM
  Dave Farley

Pratiquer

- Code Retreat: https://www.coderetreat.org
- Meetup: https://www.meetup.com/fr-FR/software-craftsmanship-lyon/
- Cyber dojo: http://cyber-dojo.org/
- CodingDojo: http://codingdojo.org/kata/