

Test Driven Development

L'idée

- Commencer par écrire les tests
- Faire émerger le design naturellement

Etape initiale

- On découpe le problème en sous-problèmes
- On choisit le sous-problème le plus simple

Etape #1

- On écrit un test
- On imagine que le code existe déjà
- Une sorte de *pseudo code* ou de *code idéal*
-  Le test ne passe pas
-  Il ne compile carrément pas

Règle #1



Un test échouer doit, pour que coder tu puisses.

Etape intermédiaire

- On crée le code qui n'existe pas
-  Ça compile
-  Mais le test ne passe toujours pas

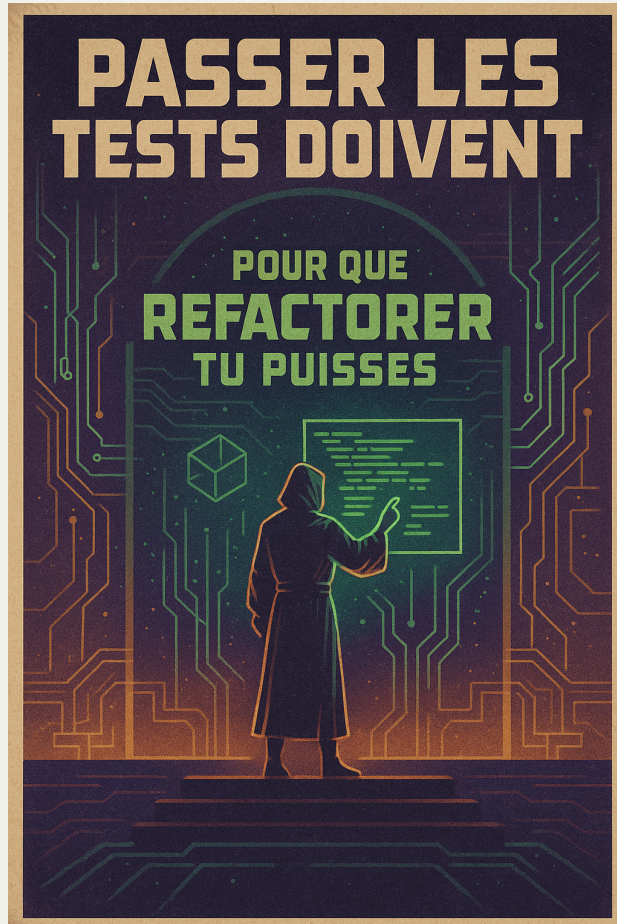
Pourquoi j'ai besoin d'un test en échec

- Un test en échec est un témoin
- C'est une preuve, une marque de progression
- Il faut vérifier qu'il échoue pour la bonne raison
- Le test ne passera que lorsqu'on aura terminé

Etape #2

- On fait une implémentation *rapidement*
- On veut faire passer le test le plus rapidement possible
- On ne prend pas de décision sur le design
- On encourage la naïveté
-  Et à la fin, le test passe
-   ...  Tous les tests passent

Règle #2



Passer les tests doivent, pour que refactorer tu puisses.

A propos du refactoring

Refactorer c'est :

- Changer l'implémentation
- Sans changer le comportement

Etape #3

- On regarde le code
 - Est-ce qu'il y a des code smells ?
 - Est-ce que c'est SOLID ?
 - ...
-  On refactor
-   ...  Les tests passent toujours

Et ensuite ?

- On prend le sous-problème suivant
- On recommence le processus

Le cycle TDD

-  Red : on écrit le test
-  Green : on implémente rapidement
-  Refactor : on fait émerger le design

Le cycle TDD

- La dernière étape n'est pas obligatoire
- Il faut parfois itérer plusieurs fois pour confirmer une intuition
- Ou parfois voir totalement autre chose émerger

Le voyage plutôt que le chemin

- Plutôt que tout prévoir à l'avance
- Le TDD s'adapte au fur et à mesure

Ressources

- **TDD by Example** de Kent Beck

Merci de votre attention

