

TDD

Qualité de Développement — R5.A.08

C. Raïevsky

2023-2024



Département Informatique

BUT Informatique 3^{ème} année

Qualité de développement - Pourquoi tester ?

Principalement pour :

- La confiance
- Les retours (*feedback*)

Mais aussi pour :

- Mesurer l'avancée
 - Peaufiner les API
 - Affiner les requis
 - Documenter les fonctionnalités
-
- Assurer la détection des erreurs et des bugs le plus tôt possible
 - Clarifier les spécifications : décider quoi tester nécessite des spécifications claires

Les tests apportent de la confiance

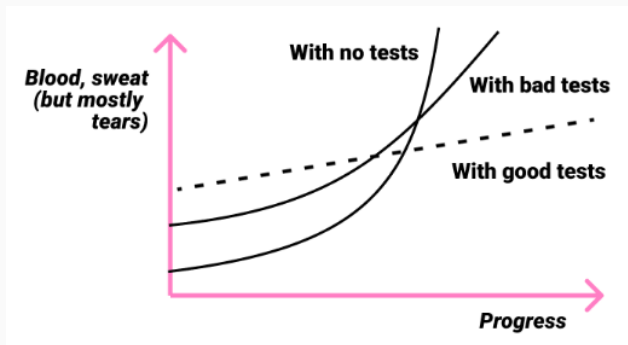
Confiance

- Détecter les erreurs et les bugs au plus tôt
- Clarifier les spécifications : décider quoi tester nécessite des spécifications claires
- Organiser le code : rendre du code testable oblige à le structurer

Gain de confiance

La confiance permet en même temps :

- Modification, Suppression, *Refactoring*
- Dormir



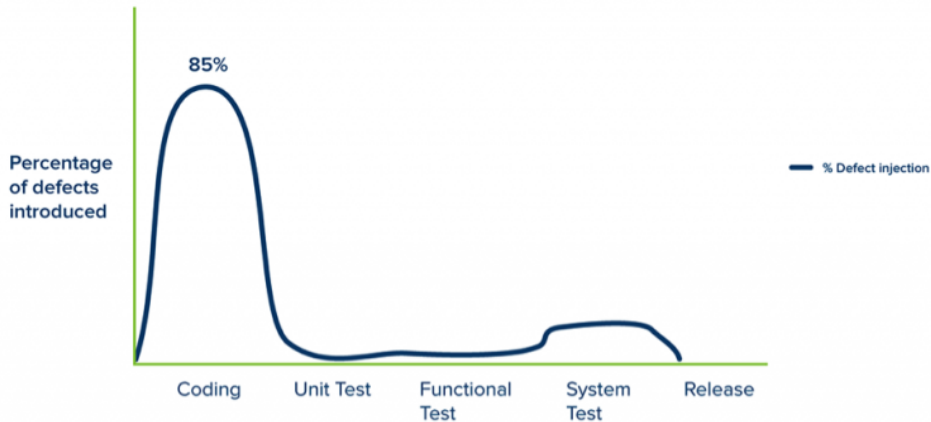
source

Les tests permettent de :

- Détecter les regressions
- Mesurer la progression
- S'assurer de la compréhension des requis
- Évaluer la faisabilité de la conception

Un code difficile à tester revèle souvent un problème de conception

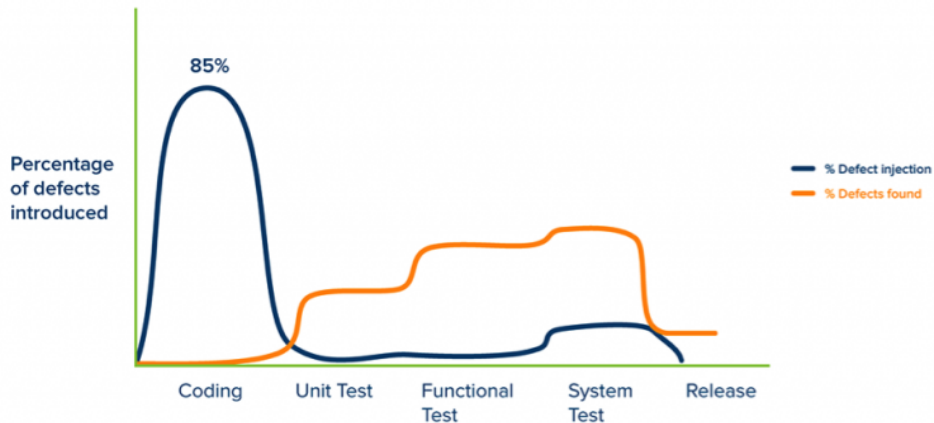
Les bugs sont créés durant le développement



Jones, Capers. *Applied Software Measurement: Global Analysis of Productivity and Quality*.

[source](#)

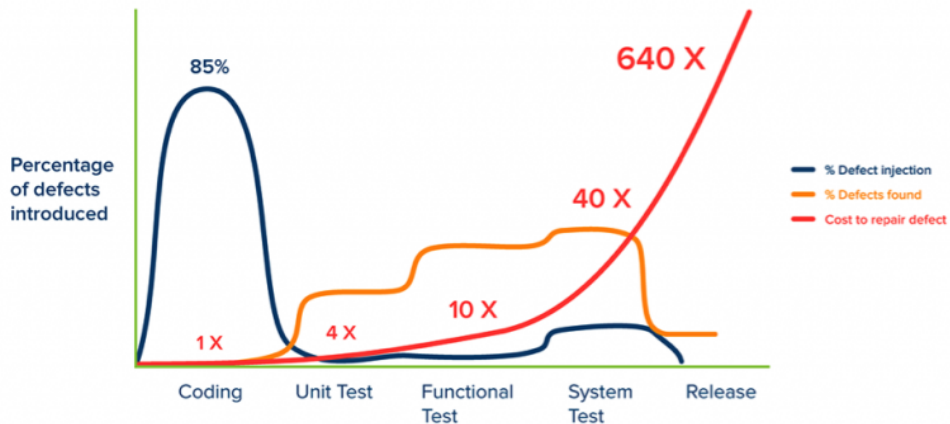
Traditionnellement les bugs sont détectés tard



Jones, Capers. *Applied Software Measurement: Global Analysis of Productivity and Quality*.

source

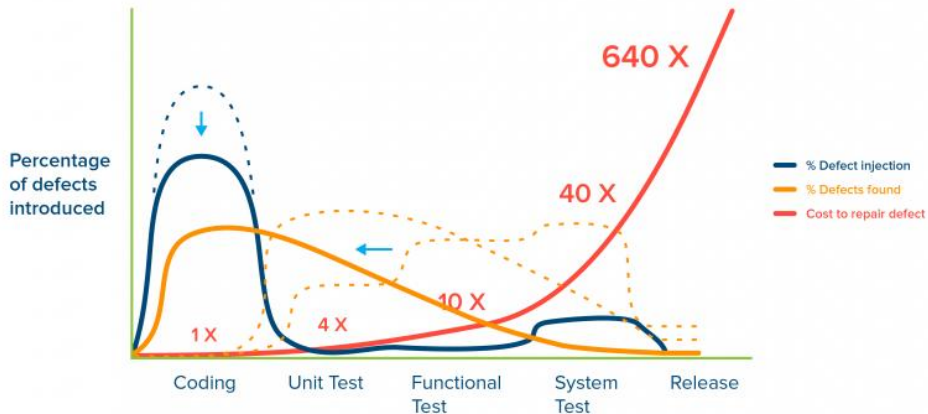
Le coût des bugs augmente très vite



Jones, Capers. *Applied Software Measurement: Global Analysis of Productivity and Quality*.

source

Décaler la détection vers la gauche - *Shift Left Approach*



Jones, Capers. *Applied Software Measurement: Global Analysis of Productivity and Quality*.

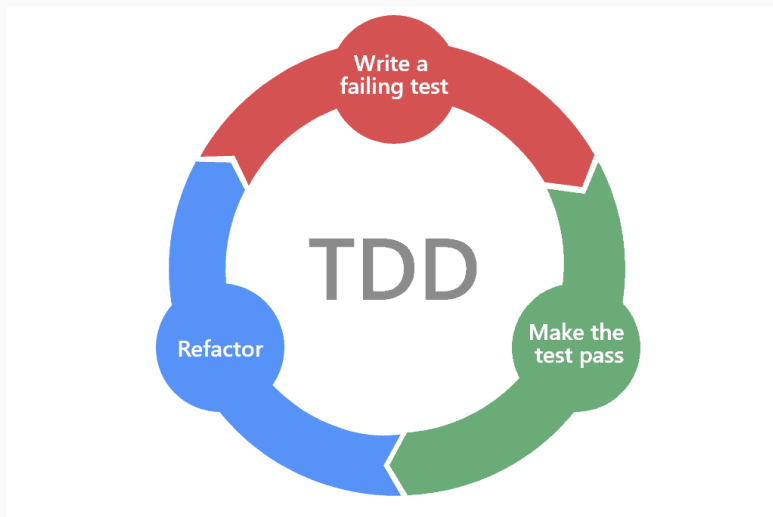
source

Test Driven Development (TDD)

Développement Dirigé par les Tests : méthode pour :

- Décaler à gauche
- Produire les tests au plus tôt
- Bénéficier de la confiance et du feedback

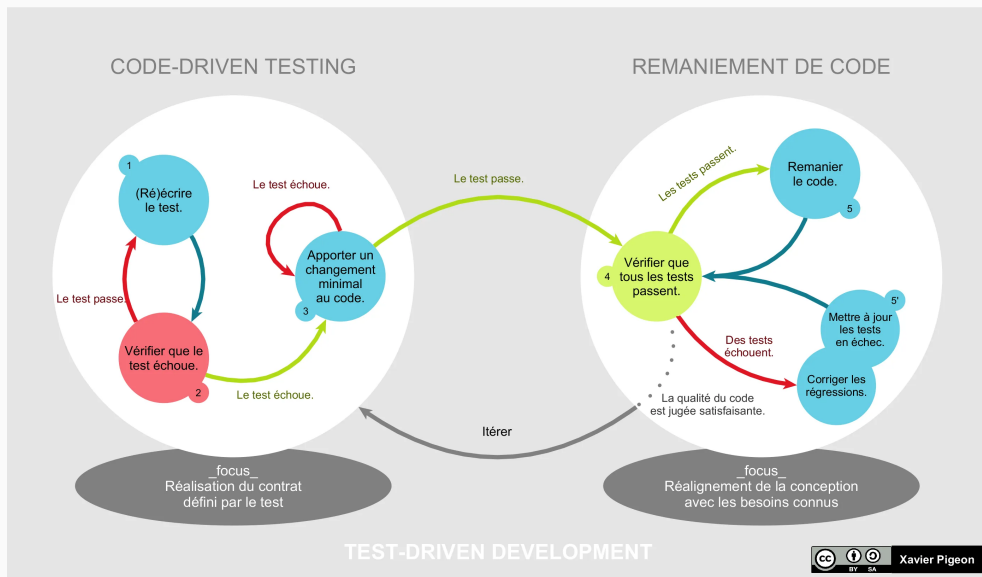
TDD - Principe de base



Les 3 règles de base

1. Vous devez écrire un test qui échoue avant de pouvoir écrire le code de production correspondant.
2. Vous devez écrire une seule assertion à la fois, qui fait échouer le test ou qui échoue à la compilation.
3. Vous devez écrire le minimum de code de production pour que l'assertion du test actuellement en échec soit satisfaite.

TDD - Vision globale



Pourquoi automatiser ?

Tous les développeurs testent, au moins une fois

Mais tester manuellement à chaque fois → **impossible**

Si on ne teste pas **tout** ou pas **régulièrement** :

- Pas de confiance
- Pas de feedback