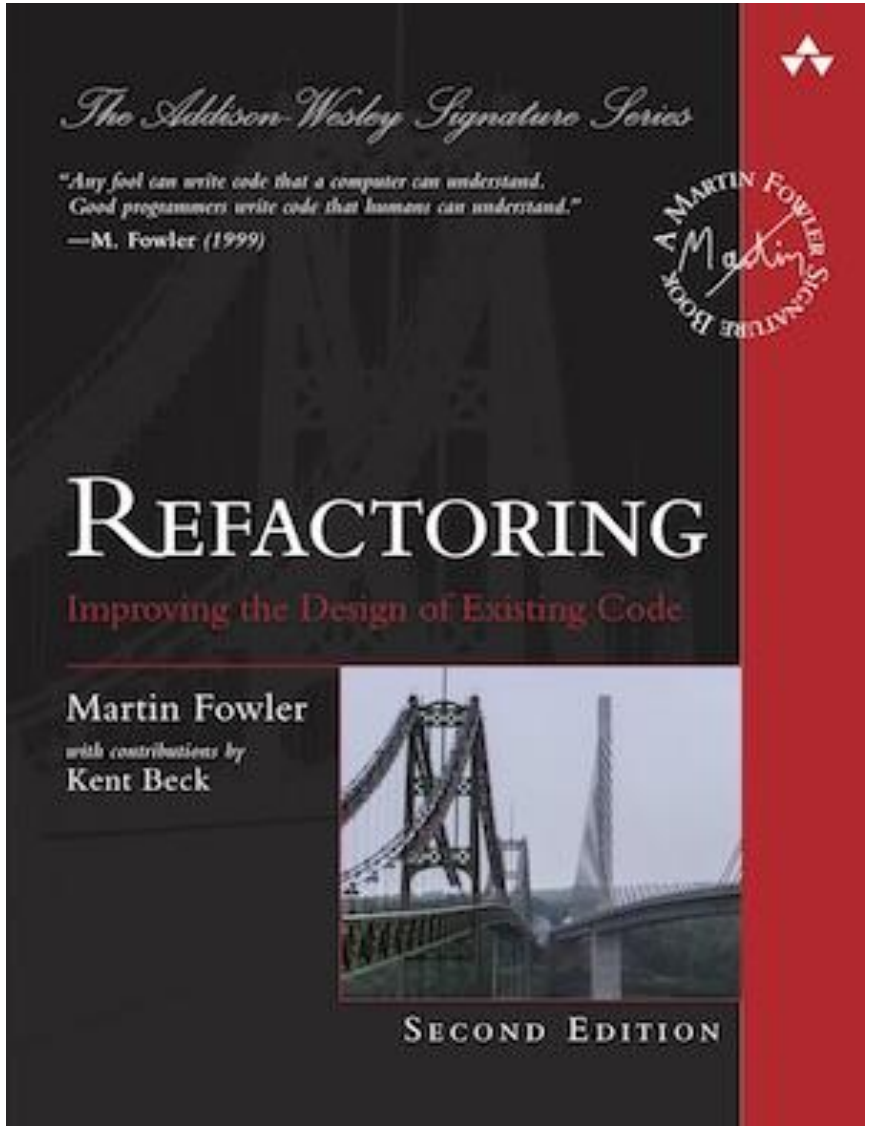



# Refactoring

# Règles d’Ouverture

- Liberté de parole
  - Tu peux poser des questions, te tromper, challenger les idées.
- Pas de 'chef’
  - Je facilite, mais la valeur vient de l’échange collectif.
- Les deux pieds
  - Si ce n’est pas utile pour toi → tu es libre de partir.

# Pourquoi ?

- Rendre le code testable : tests = filet de sécurité.
- Lisibilité
- Maintenabilité
- Réduire le risque de régressions

# Qu’est-ce qui rend le refacto compliqué ?

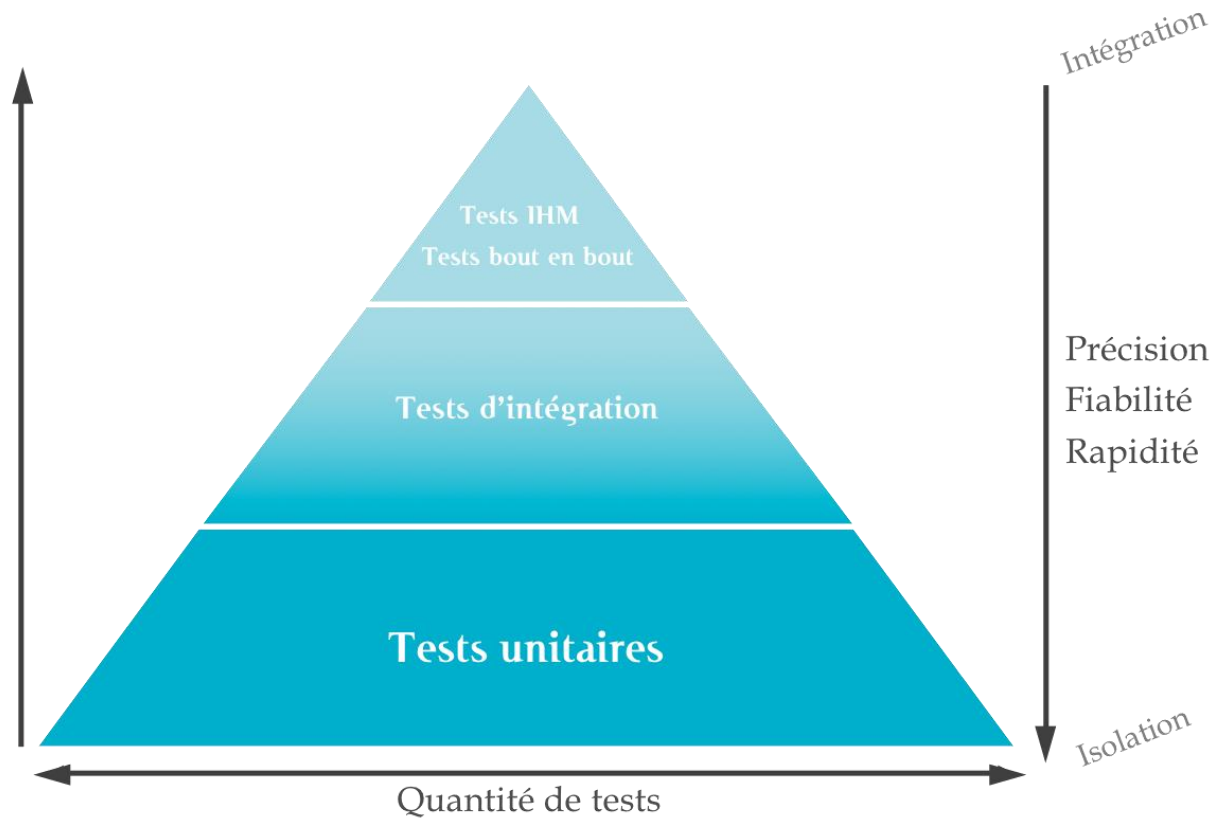
- Architecture & Couplage : schéma de dépendances “spaghetti”.
- Non respect de SOLID :
  - SRP (**Single Responsability Principle**) : Une classe, module, fonction, ne doit avoir qu’une seule raison de changer
  - DIP (**Dependency Inversion Principle**) :dépendre d’abstractions, pas de détails
- Manque de tests : pas de filet de sécurité → peur de toucher au code.

C’est quoi un test ?

C’est quoi un bon test ?

# Comment fait-on sur du legacy ?

- Tests E2E : capturer le comportement global, filet de sécurité haut niveau.
- Golden Master : sécuriser le comportement actuel
- Faire confiance à l'IDE : rename, extract method, move class.



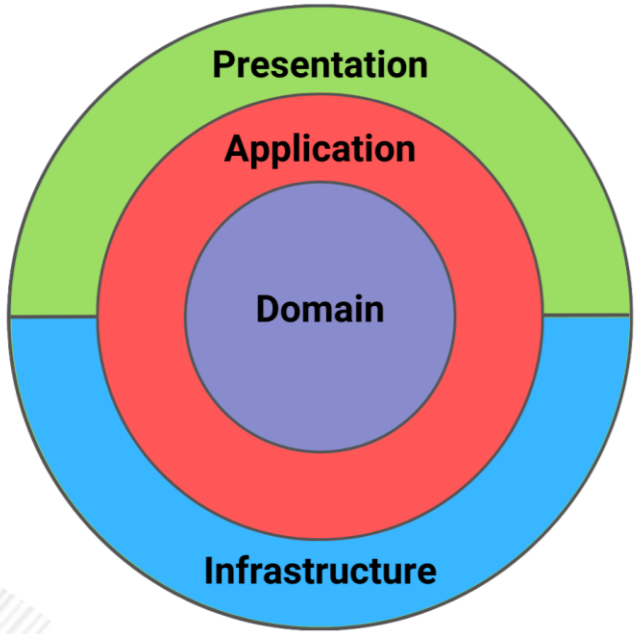
# Quand est-ce qu’on refacto ?

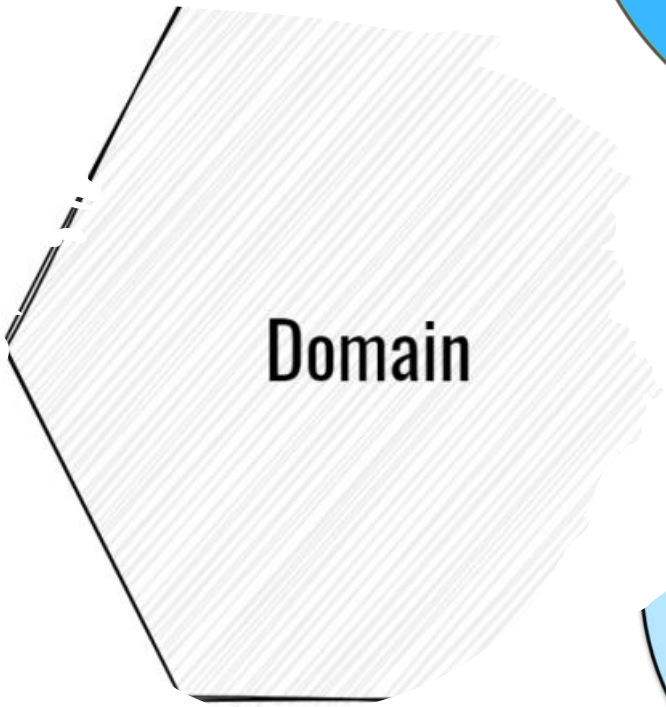
- Hotspot : code **souvent modifié** → forte valeur de le rendre propre.
- Zones avec **régressions fréquentes** → le refacto réduit les bugs récurrents.
- Code **extrêmement complexe** → la complexité elle-même devient un risque.

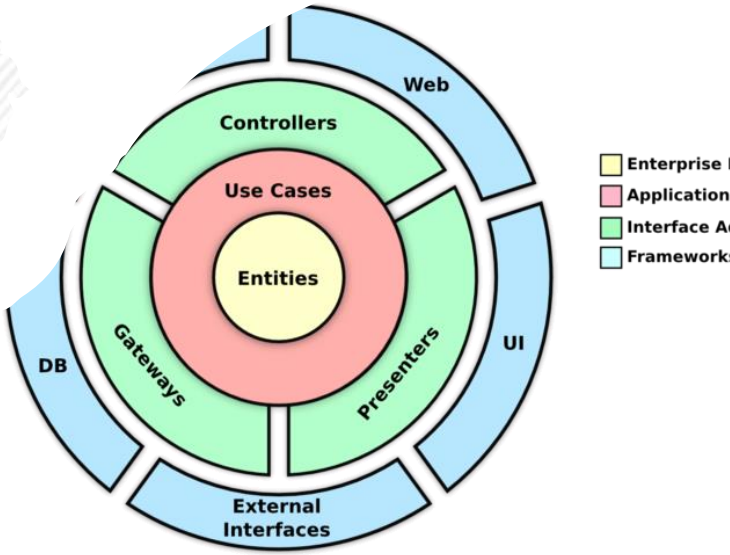
# Stratégies de refactoring

- **Nouveau code** : Clean/hexa/onion Architecture + TDD
  - → Refactoring sans risques grâce aux tests.
- **Code historique (legacy)** :
  - Golden Master (démo à faire)  ou Tests E2E
    → Filet de sécurité indispensable.
- **Pair programming**
  - Un regard extérieur limite les erreurs
  - Permet d'oser avec plus de confiance

⚠️ Pas de refactoring sans tests.

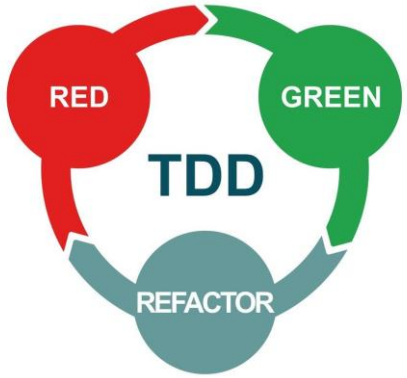


# KATA : Gilded Rose

- Mob programming
- Timebox : 1h30 Max
- Mieux vaut arrêter en plein milieu que dépasser le temps.
- L’objectif = apprendre ensemble, pas produire du “beau code” parfait
- “sans clavier” et sans copier/coller → montrer la puissance des refactors auto.




# Next Steps

- 📌 **Tests unitaires** – fondamentaux (structure, outils, isolation)
- 📌 Découvrir le **TDD** :
Cycle **Red → Green → Refactor** pour concevoir avec des tests.
- 📌 Approfondir le **refactoring avec I/O** :
Isoler lectures/écritures (fichiers, DB, API) pour rendre le code testable.
- 📌 Objectif : aller vers une architecture plus découplée et robuste.

# Rétro

# Biblio

- https://martinfowler.com/books/refactoring.html
- https://www.dunod.com/sciences-techniques/software-craft-tdd-clean-code-et-autres-pratiques-essentielles-0
- Arnaud Lemaire : https://www.youtube.com/watch?v=RZJJLQPHgvw&t=3405s&ab_channel=BreizhCamp
- https://www.youtube.com/watch?v=QFtr0yRpvPk&ab_channel=BreizhJUG