Bonnes pratiques de tests

Tests automatisés - Module 3

Comment automatiser les tests?

Comment coder pour faciliter les tests?

Quelles sont les bonnes pratiques à appliquer?

Exemples d’automatisation

Comment coder pour faciliter les tests? Limiter les chemins algorithmiques

- Comme on l'a vu sur la complexité cyclomatique, plus un morceau de code peut avoir de chemins algorithmiques, plus il faut écrire de tests pour les couvrir.
- Il faut donc limiter le nombre de chemins que peut prendre l'exécution du code.

Comment coder pour faciliter les tests? Limiter les chemins algorithmiques

- Pour cela, on va chercher à utiliser des design patterns comme strategy par exemple.

  Si on a un switch/case sur une même variable à plusieurs endroits dans le code, on va chercher à instancier un objet qui sera en mesure d'effectuer les différents traitements en fonction de la valeur de la variable.

Comment coder pour faciliter les tests? Limiter les chemins algorithmiques

- Surtout, on va chercher à respecter le principe de responsabilité unique, et chercher à avoir des fonctions qui ne font qu'une seule chose.

=> Respect du SoC « Séparation of Concerns ».

Par exemple, on ne fait pas un traitement dans la même classe que l'enregistrement en BDD ou la mise en forme dans un tableau excel.

Comment coder pour faciliter les tests ? Maximiser le déterminisme

- De la même manière que pour le nombre de chemins algorithmiques, si un code dépend de contextes très différents, il sera plus dur à tester.
- Par exemple, un test qui peut échouer en fonction de données ramenées d’une source externe n’est pas fiable.
- Un autre classique est un test dont le code dépend de l’heure à laquelle

Comment coder pour faciliter les tests ? Maximiser le déterminisme

- On va donc éviter les choses suivantes dans le code :
  - Récupérer l'heure depuis le système. On passe par un provider.
  - Récupérer des données depuis un fichier, une BDD, une API dans une méthode qui effectue le traitement… (Bonus pour les connectionString en dur dans le code)
  - Écrire directement sur le filesystem à un chemin donné (la CI ne tourne peut être pas sur le même OS que votre PC de dev)
  - L'aléatoire.

Comment coder pour faciliter les tests ? Maximiser le déterminisme

- On va donc éviter les choses suivantes dans les tests:
  - Partager un état entre les tests.

    Avoir un test qui insère une data et un autre qui la récupère peut ressembler à une bonne idée … Jusqu'au jour où la suite de tests est trop lente et qu'il faut paralléliser l'exécution ! (au plus tard)
  - Reposer sur l'environnement. (Variables d'environnement par exemple)

Quelles sont les bonnes pratiques à appliquer? Quelques astuces

- **Tester les cas “limites” de façon précise**
- **Utiliser des valeurs proches des limites gérées par les types utilisés (erreurs de débordement et de débordement des types de données comme un entier)**
- **Tester les exceptions et les erreurs**
- **Tester des arguments erronés**

Quelles sont les bonnes pratiques à appliquer? Quelques astuces

<u>Positive Testing :</u> “Cas passant”

Tester que l’application fonctionne dans un contexte valide (login OK)

<u>Negative Testing :</u> “Cas non passant”

Tester que l’application échoue dans un contexte invalide, techniquement (exceptions, erreurs, arguments erronés), mais aussi d’un point de vue métier (mauvais login inséré)

Quelles sont les bonnes pratiques à appliquer?
Les principes **F.I.R.S.T.**

F for FAST : les tests doivent être rapides

I for Isolated: un test doit isoler les erreurs, les bugs

R for Repeatable : résultat identique quel que soit l'ordre d'exécution

S for Self-Validating : aucune action manuelle ne doit être requise.

T for Timely : écrits **avant** le code de production

Quelles sont les bonnes pratiques à appliquer?

DAMP: être compris plutot que concis

DAMP : Descriptive And Meaningful Phrases

Un bon test doit pouvoir être lu et compris presque aussi facilement qu'un langage « humain ». Pour cela, on va utiliser au mieux le nommage des classes de tests, des tests eux-même, et des méthodes qu'ils appellent.

Il ne faut pas avoir peur des noms longs, ou de créer des méthodes juste pour exprimer une intention via leur nom.

Quelles sont les bonnes pratiques à appliquer?
DAMP: être compris plutot que concis

Organisation d'un test :

- **Méthode « AAA » : Arrange, Act, Assert**
  - **Souvent utilisé dans les tests unitaires,**
- **Méthode « Given, When, Then »**
  - **Utilisé principalement en Behaviour Driven Design (Gherkin)**

Quelles sont les bonnes pratiques à appliquer?
Quels types de tests, dans quelle quantité ?

La pyramide des tests :

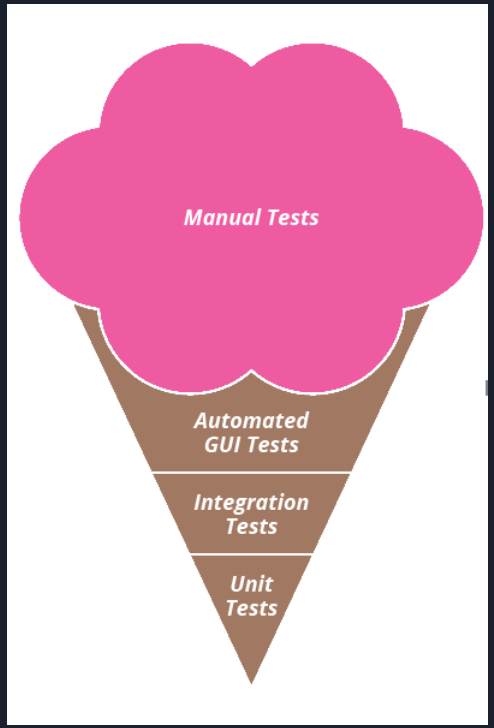
C’est une représentation schématique où on représente dans un repère le volume de tests liés à un projet. En haut, les tests les plus intégrés et donc les plus lents et les plus chers. Plus on descend, plus les tests sont isolés et rapides.

Si on a un maximum de tests rapides et efficients, et donc on aura une forme de pyramide.

2

Quelles sont les bonnes pratiques à appliquer?

Quels types de tests, dans quelle quantité ?



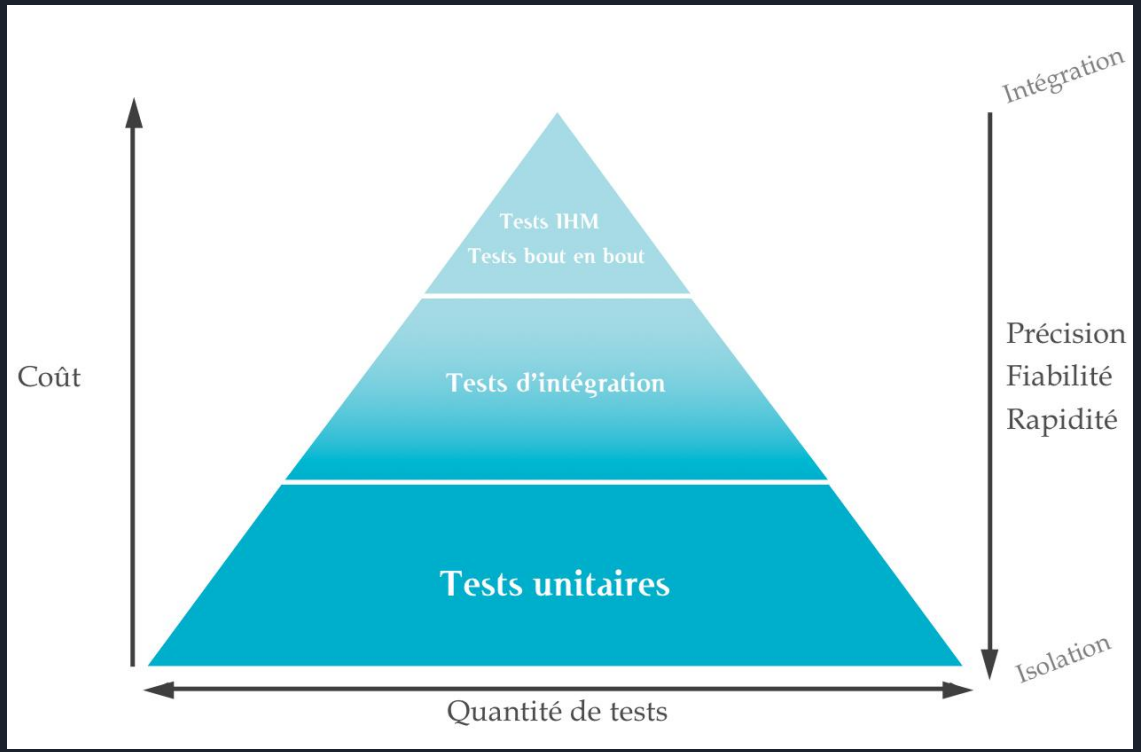
Méthode “traditionnelle”

La-rache®

2

Quelles sont les bonnes pratiques à appliquer?

Quels types de tests, dans quelle quantité ?

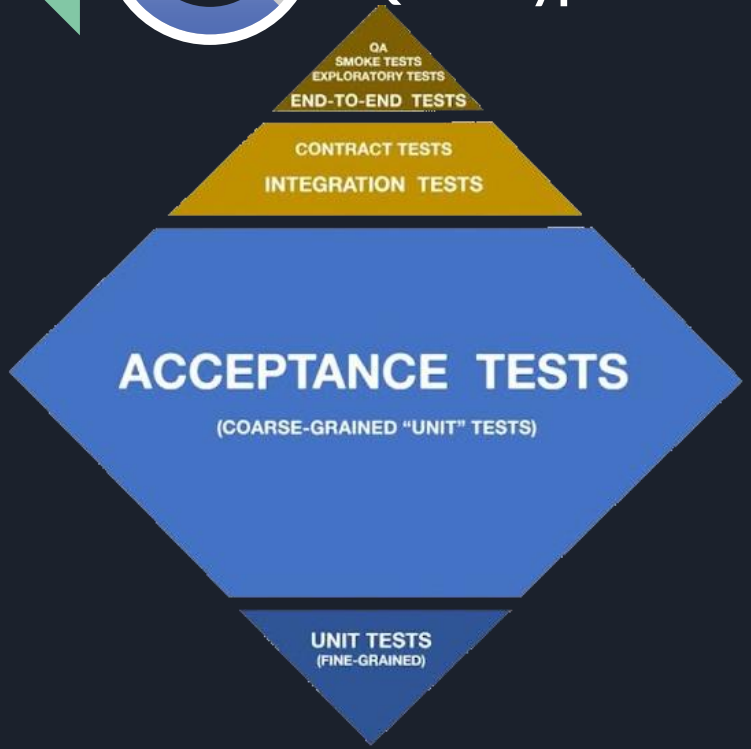


Méthode Agile

2

Quelles sont les bonnes pratiques à appliquer?

Quels types de tests, dans quelle quantité ?



« Outside-in Diamond »
de T. Pierrain

Exemples d'automatisation des tests
Refactorisation par la méthode "Golden Master"

Étapes:

1. Lancer le programme dans de nombreux cas, et sérialiser les données de départ, et le résultat du calcul
2. Implémenter le traitement dans sa nouvelle implémentation
3. Coder un test qui appelle nos nouveaux objets avec les données initiales, et vérifier le résultat.
4. Extraire le comportement de la classe de base et remplacer les appels aux anciennes méthodes par les nouvelles.

Exemples d’automatisation des tests
Refactorisation et corrections de bugs sur un code existant

Premières Étapes: Refactoring

1. Comprendre le code, le rendre testable grâce aux abstractions, le tester (manuellement ou non).
2. Coder des tests qui valident le comportement existant
3. Séparer en méthodes et en classes les différentes parties du traitement (respect de SRP, technique Extract Method)

Exemples d'automatisation des tests
Refactorisation et corrections de bugs sur un code existant

Étapes suivantes : Corrections et ajout de fonctionnalités

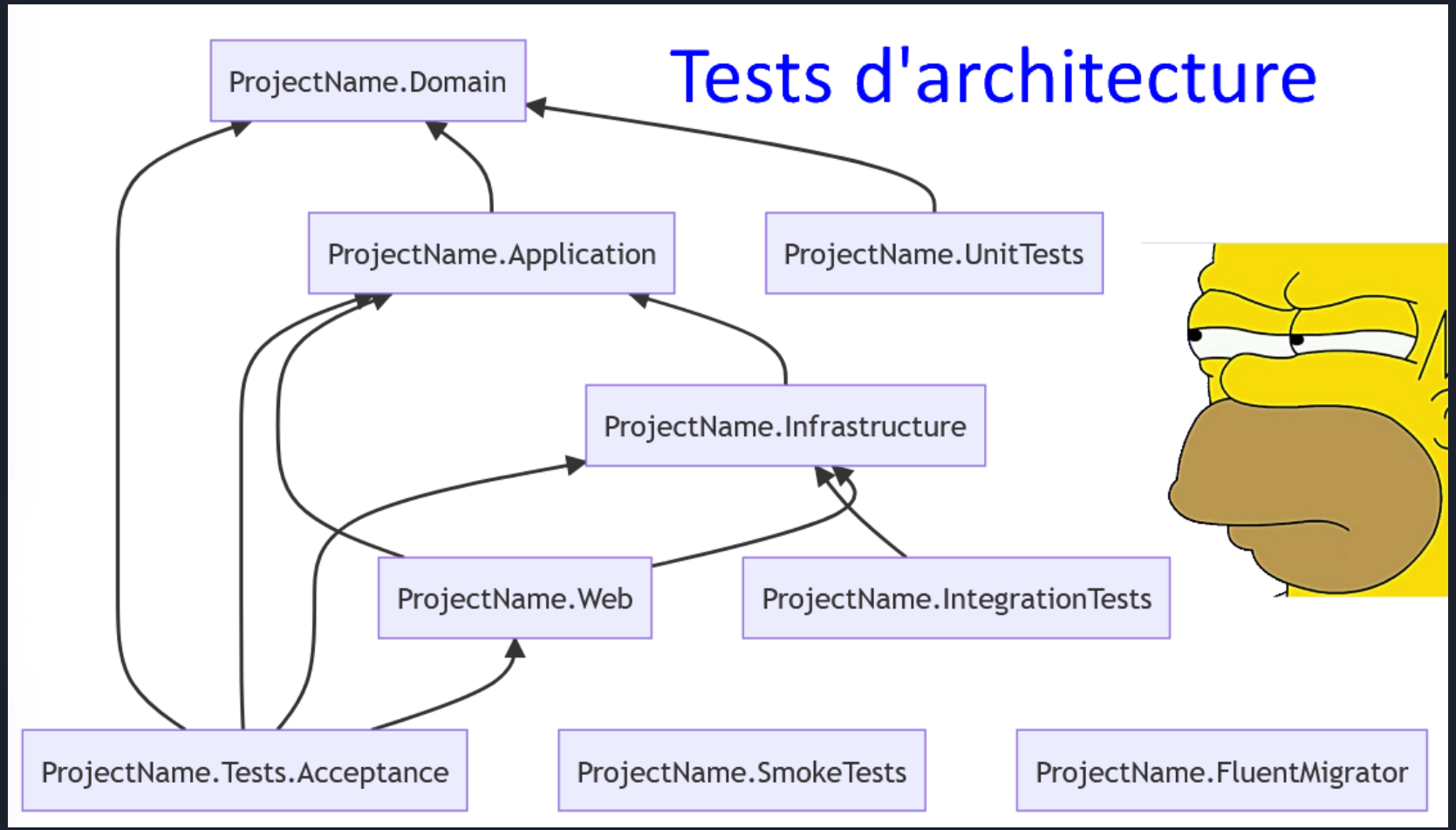
4. Coder des tests qui reproduisent un bug, en ajoutant un Assert qui doit échouer (car il vérifie le comportement souhaité)
5. Coder la modification qui valide le test précédent, sans "casser" les autres
6. Refactorer le code pour avoir un design de qualité, en utilisant les tests pour s'assurer que le comportement souhaité est toujours celui qui est en place.

Exemples d’automatisation des tests
Tests d’architecture

On peut vérifier le code au niveau algorithmique, mais on peut également utiliser certains outils pour tester que les standards d’architecture fixés par l’équipe sont bien respectés.

Par exemple, dans l’écosystème .NET, on peut utiliser le paquet nuget ArchUnitNET.

Tests d'architecture



La temporalité des tests

Quand coder les tests?

Quelles sont les bonnes pratiques à appliquer?

Exemples d’automatisation

Quand écrire un test ?

Test After : Après avoir écrit le code

- On garantit uniquement que le code fonctionne bien comme on a pu l'observer, bugs inclus.
- On capture le fonctionnement, mais on ne garantit pas que l'intention est explicitée par le test. On a souvent un biais de confirmation.
- N'évite pas le code mort, n'aide pas à la qualité du code.
- Parfois difficile car le code n'a pas été pensé pour être testable.
- On n'obtient de la valeur que pour la non-régression.

Quand écrire un test ?

Test First: Après avoir fait la conception de l’architecture

- Nécessite de faire toute la conception, et qu’elle soit valide.
- Utilisé quand un architecte fait toute la conception, et quelqu’un d’autre fait l’implémentation.
- Peut servir dans le cas d’un développement “outsourcé” pour garantir que l’appli correspond à l’implémentation souhaitée.

Quand écrire un test ?

Test Driven Development : Avant le code de production en 3 étapes

Ecrire un test pour lequel le code n'est pas encore écrit

Ecrire le minimum de code pour faire passer le test

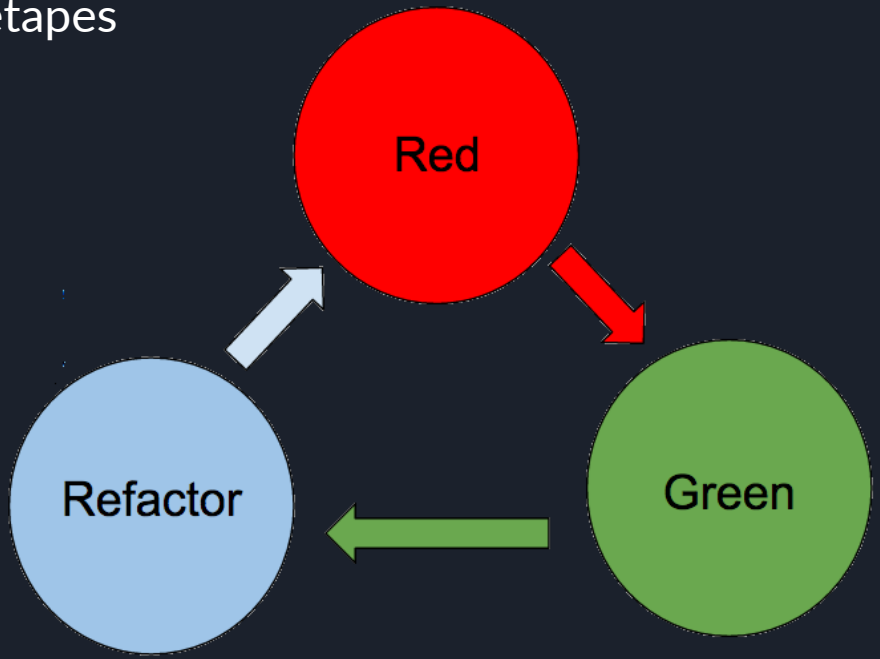
Réusiner le code pour avoir un code de qualité

Comment automatiser les tests ?

Quand écrire un test ?

3

Test Driven Development : Avant le code de production en 3 étapes



Comment automatiser les tests ?

Quand écrire un test ?

Test Driven Development : Avant le code de production

Les + :

- Le code émerge des contraintes du métier,
- Le cycle de refactoring garantit une qualité constante,
- Couverture de code par les tests proche de 100%

Les - :

- Courbe d’apprentissage,
- Nécessite de la discipline

Comment automatiser les tests ?

ATTENTION

Les tests automatisés n’ont de valeur que si on les exécute **régulièrement**