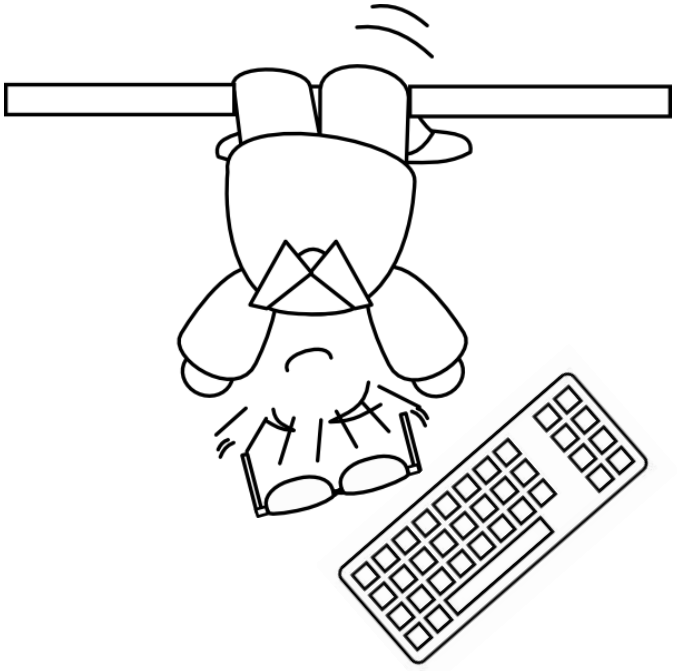
Coder à l'envers

pour

penser à l'endroit

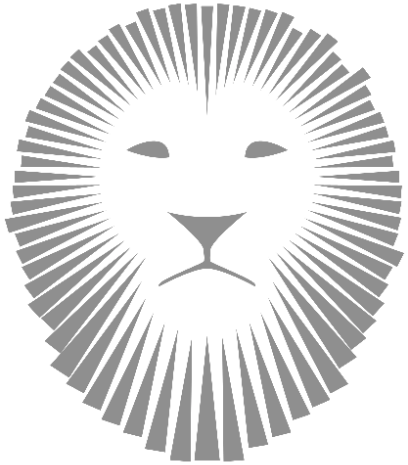
2ème partie



Approfondissement

du TDD avec le

Outside-In TDD



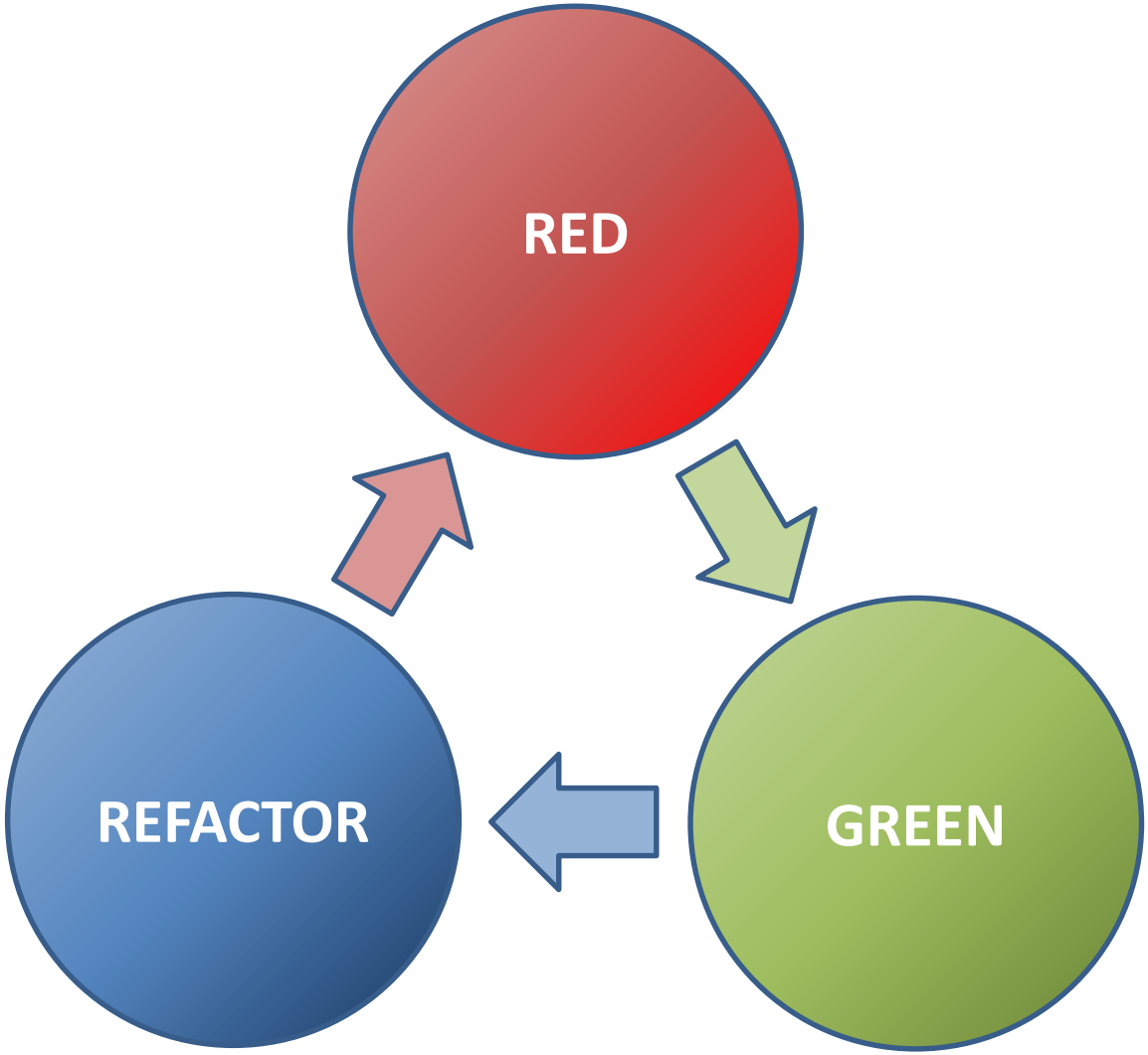
Khris

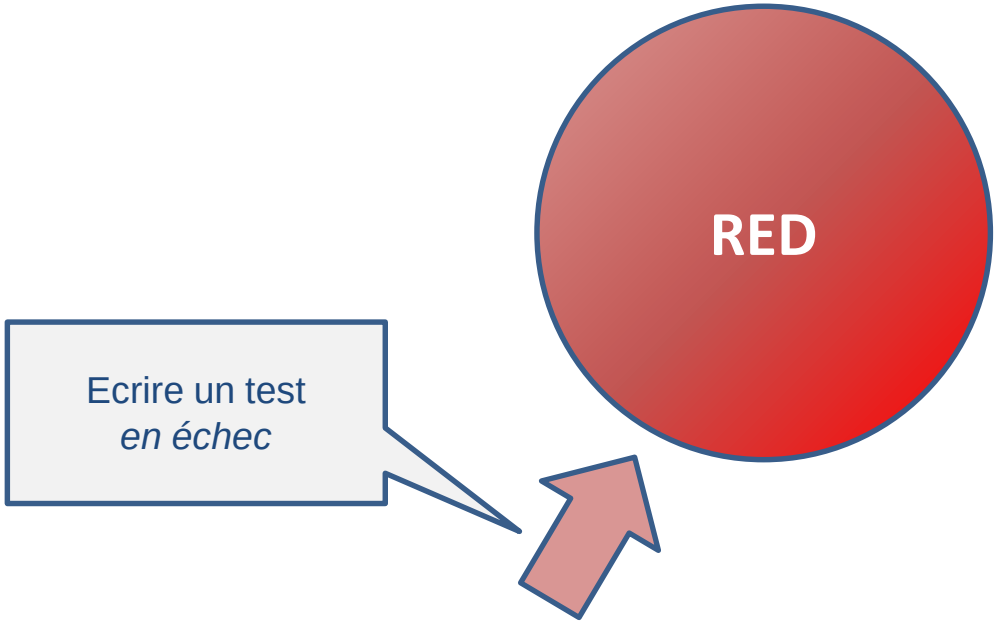
lyontechhub.slack.com

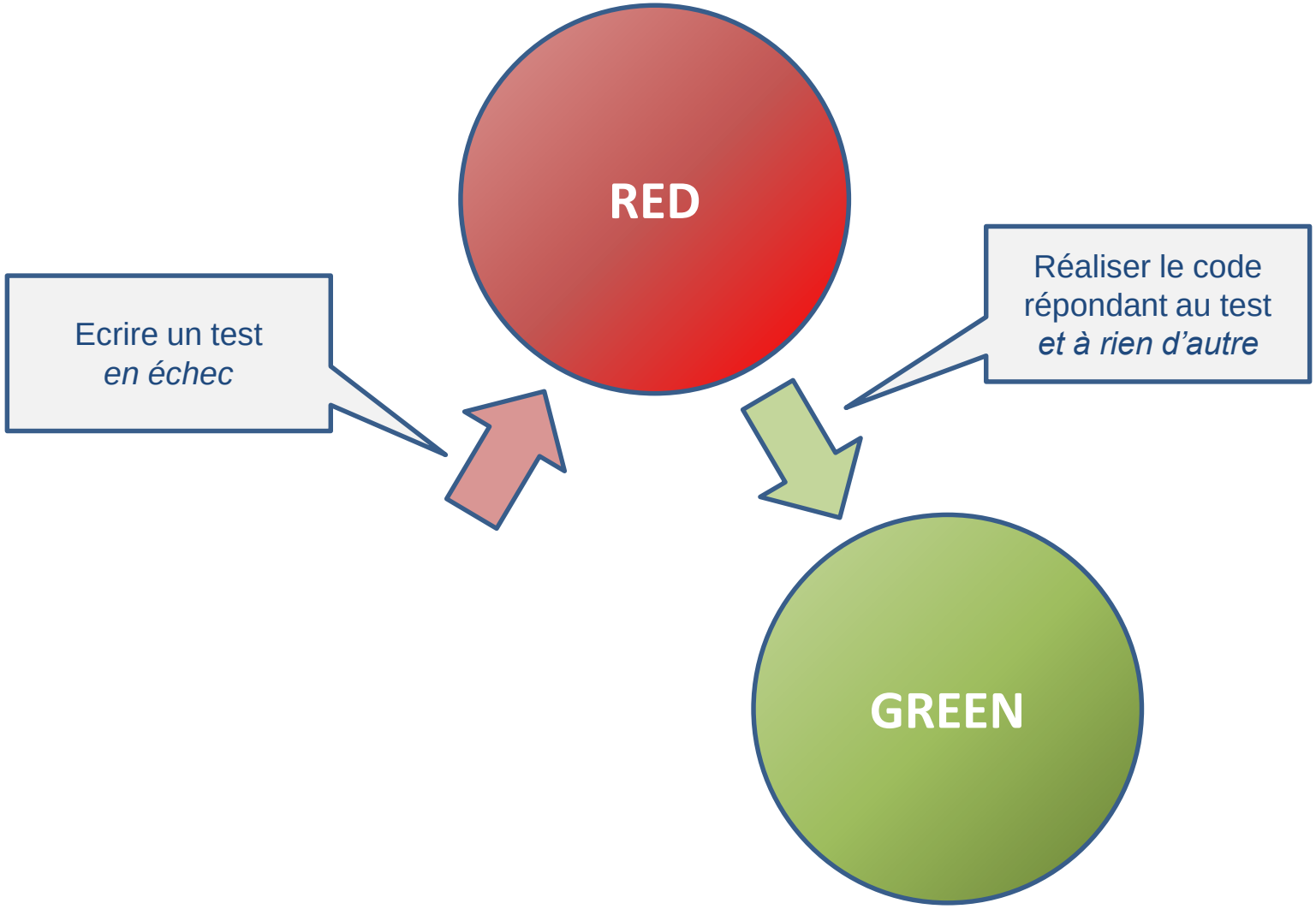
Le TDD ...

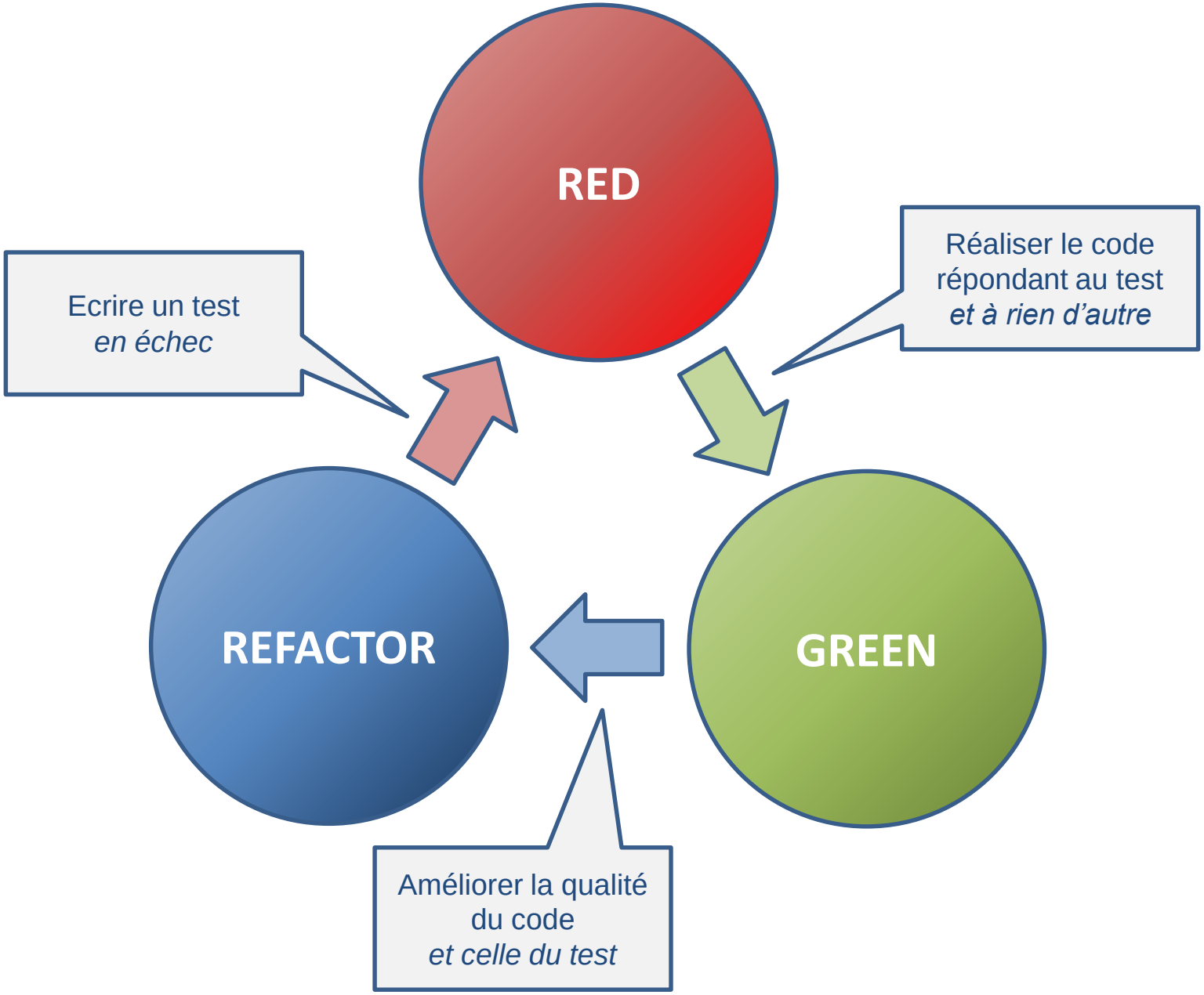
... c’est quoi ?

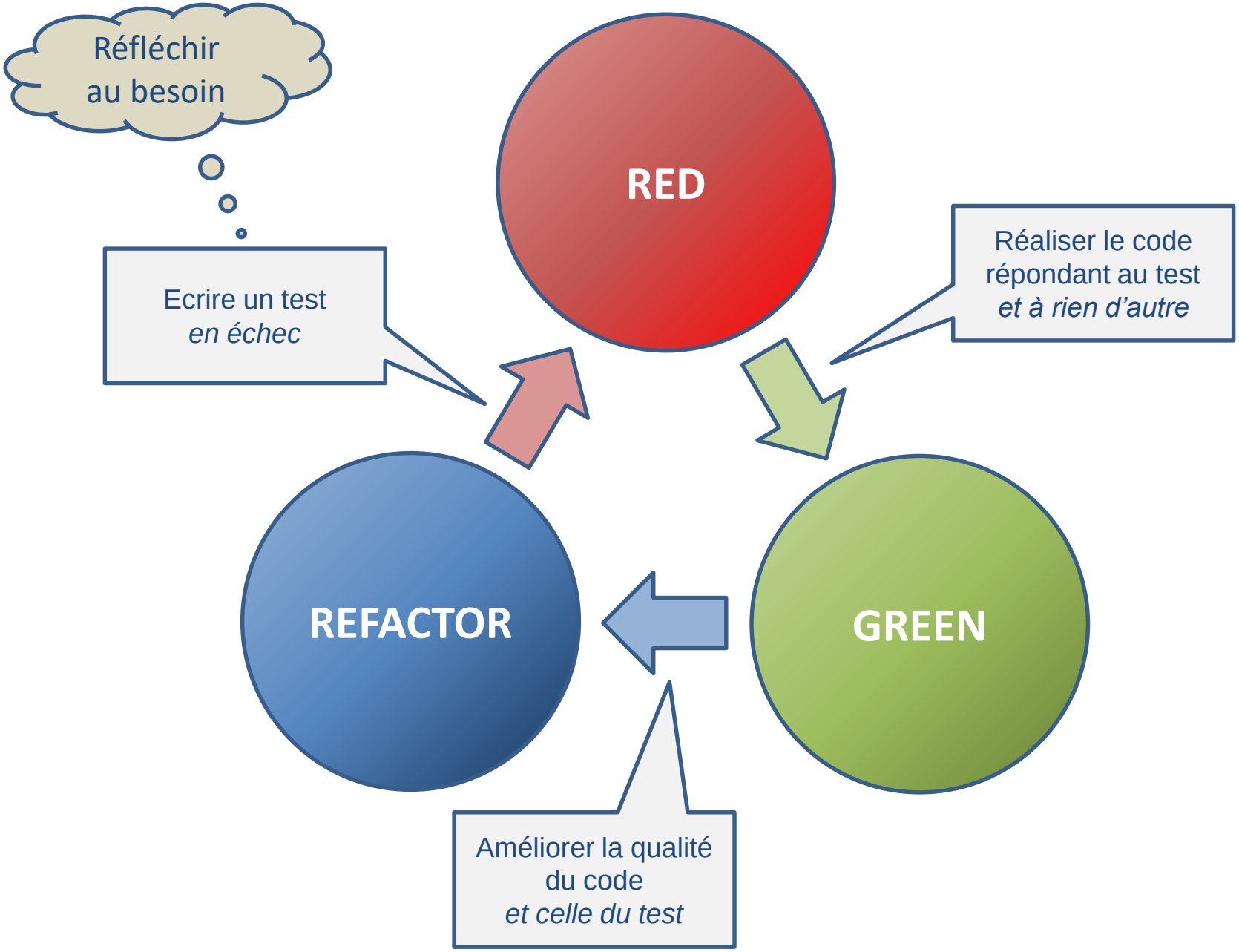
Le TDD « classique »

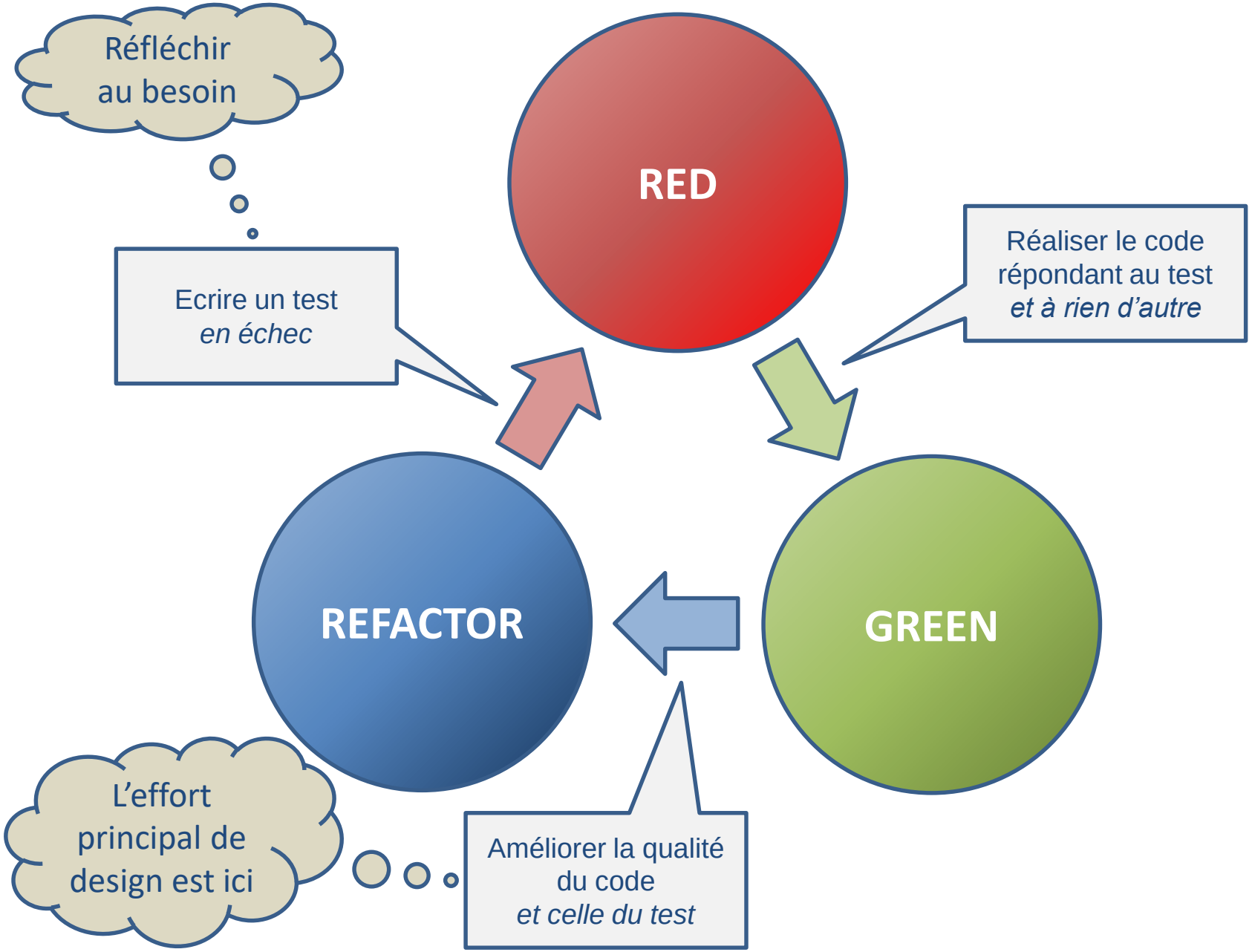

RED
GREEN
REFACTOR


RED
Ecrire un test
en échec

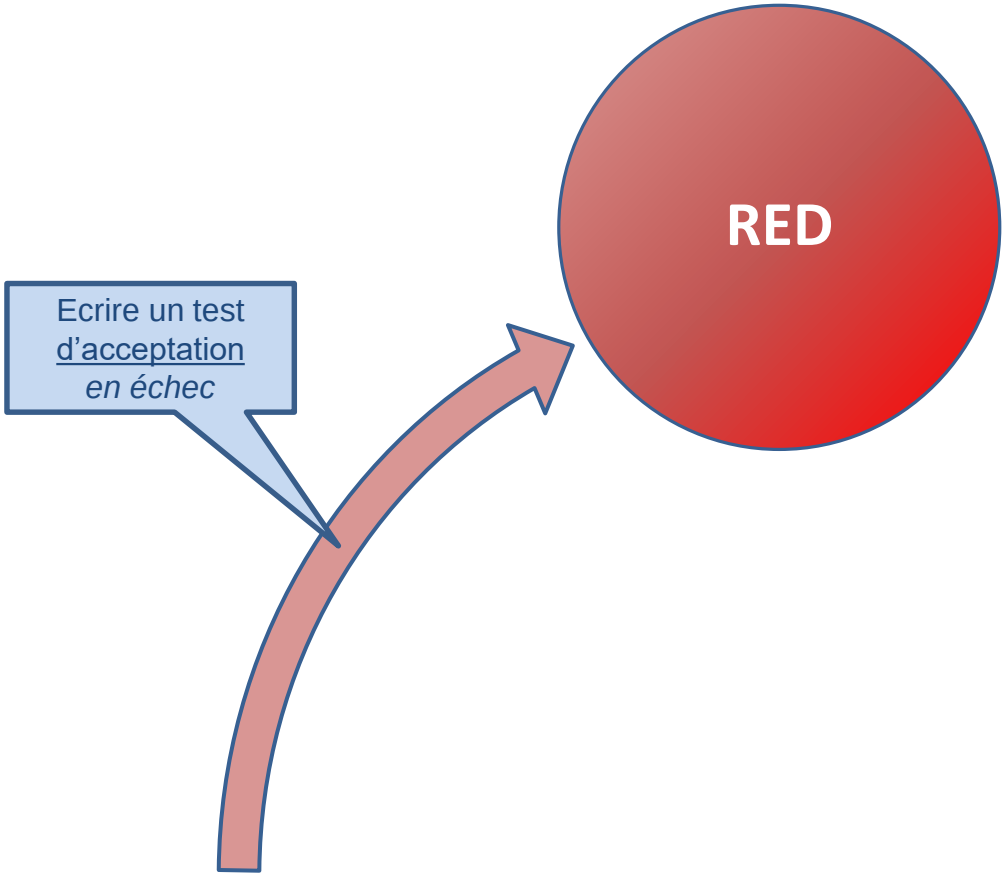

RED
Ecrire un test
en échec
Réaliser le code
répondant au test
et à rien d'autre
GREEN

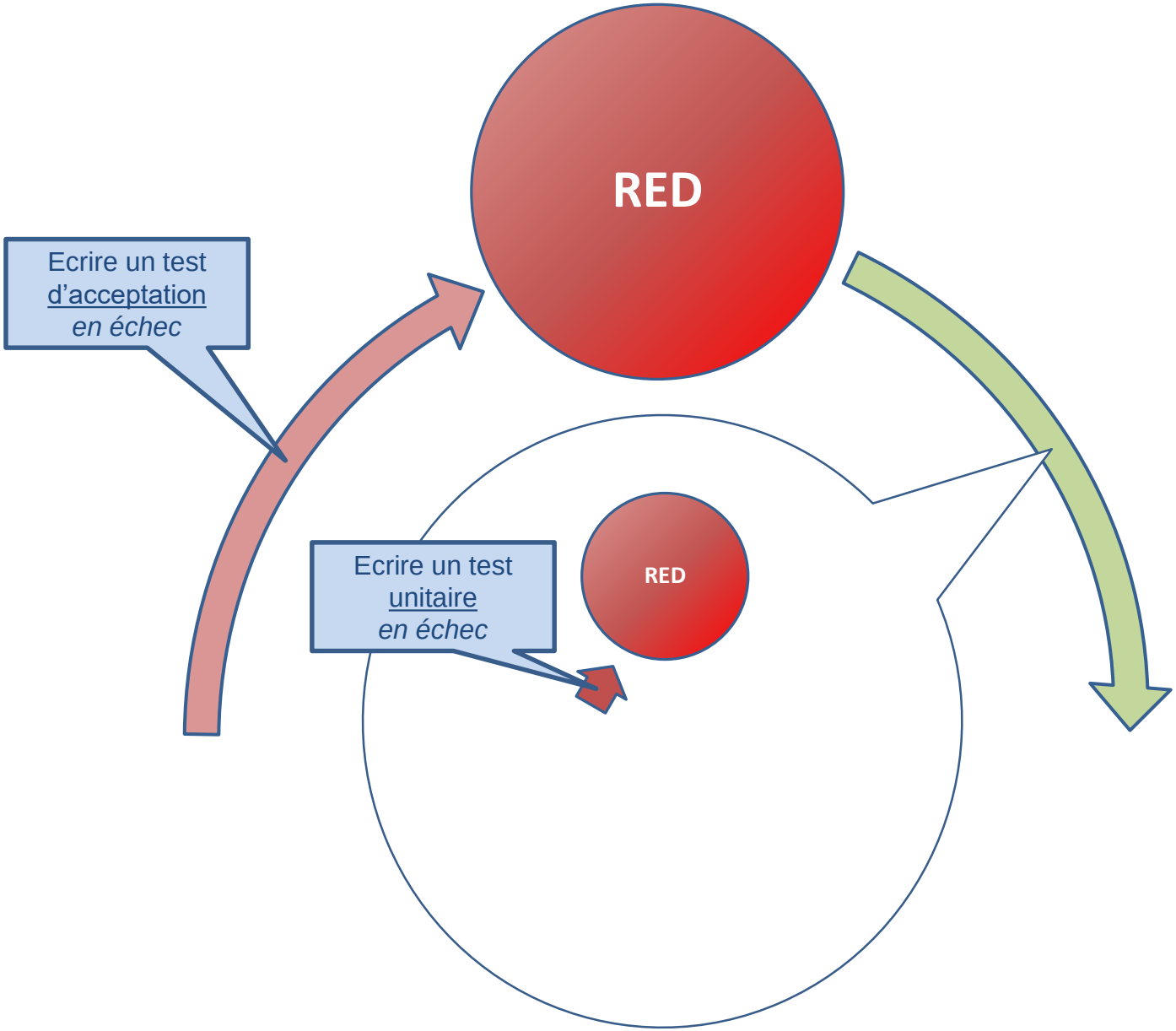

RED
Réaliser le code
répondant au test
et à rien d'autre
GREEN
Améliorer la qualité
du code
et celle du test
REFACTOR
Ecrire un test
en échec

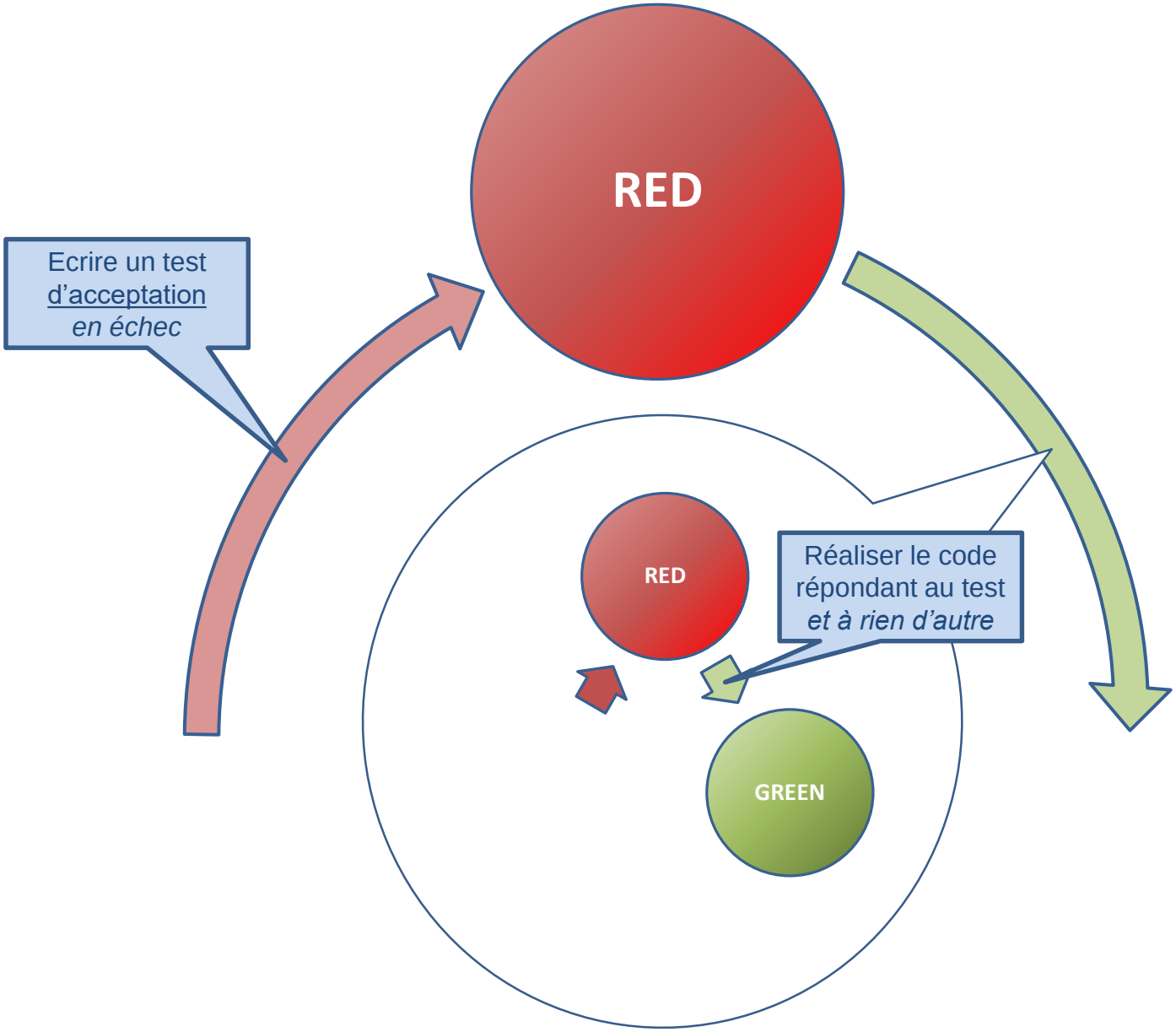

Réfléchir
au besoin
Ecrire un test
en échec
RED
Réaliser le code
répondant au test
et à rien d'autre
GREEN
Améliorer la qualité
du code
et celle du test
REFACTOR

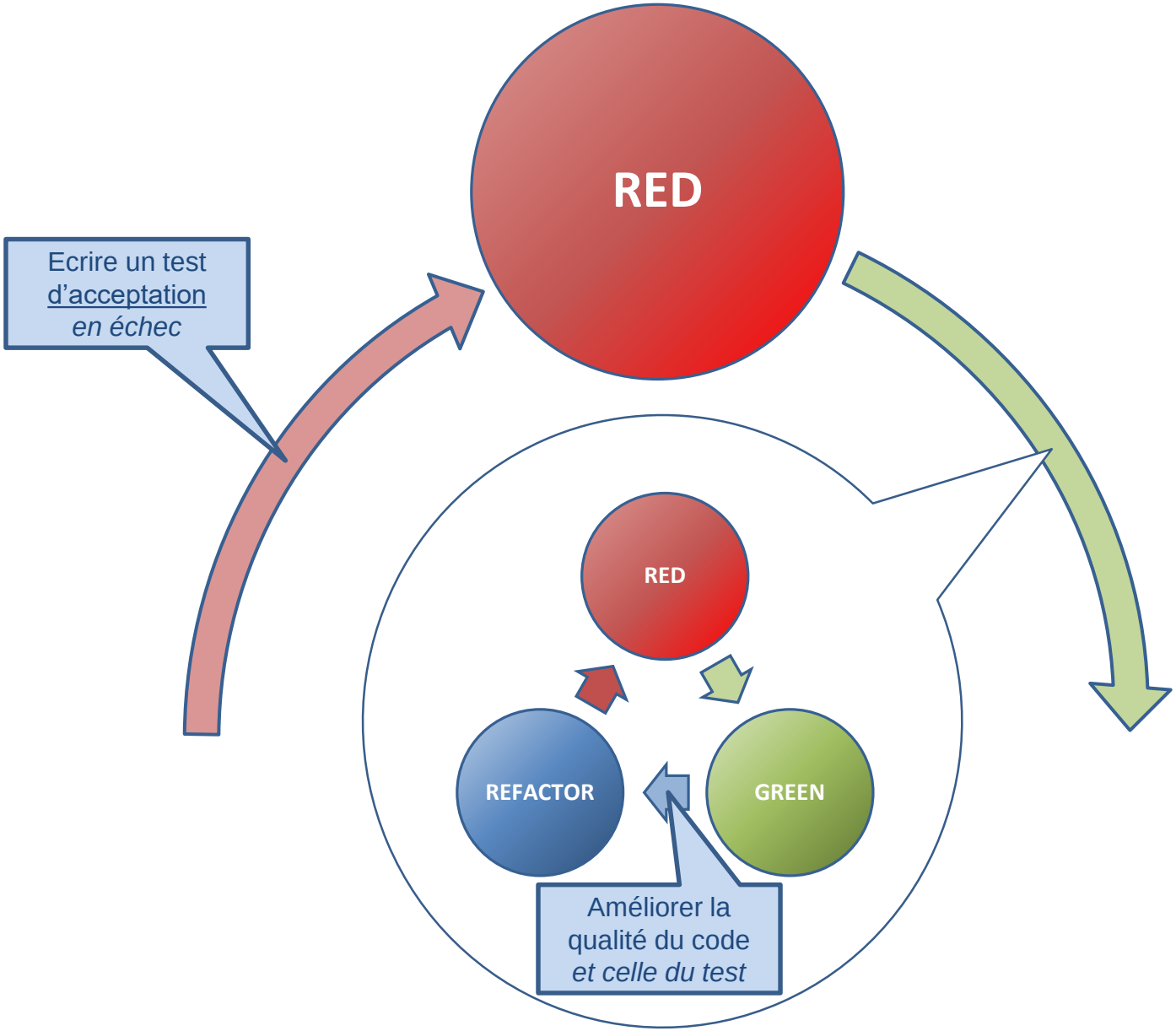

Réfléchir
au besoin
RED
Ecrire un test
en échec
Réaliser le code
répondant au test
et à rien d'autre
REFACTOR
GREEN
L'effort
principal de
design est ici
Améliorer la qualité
du code
et celle du test

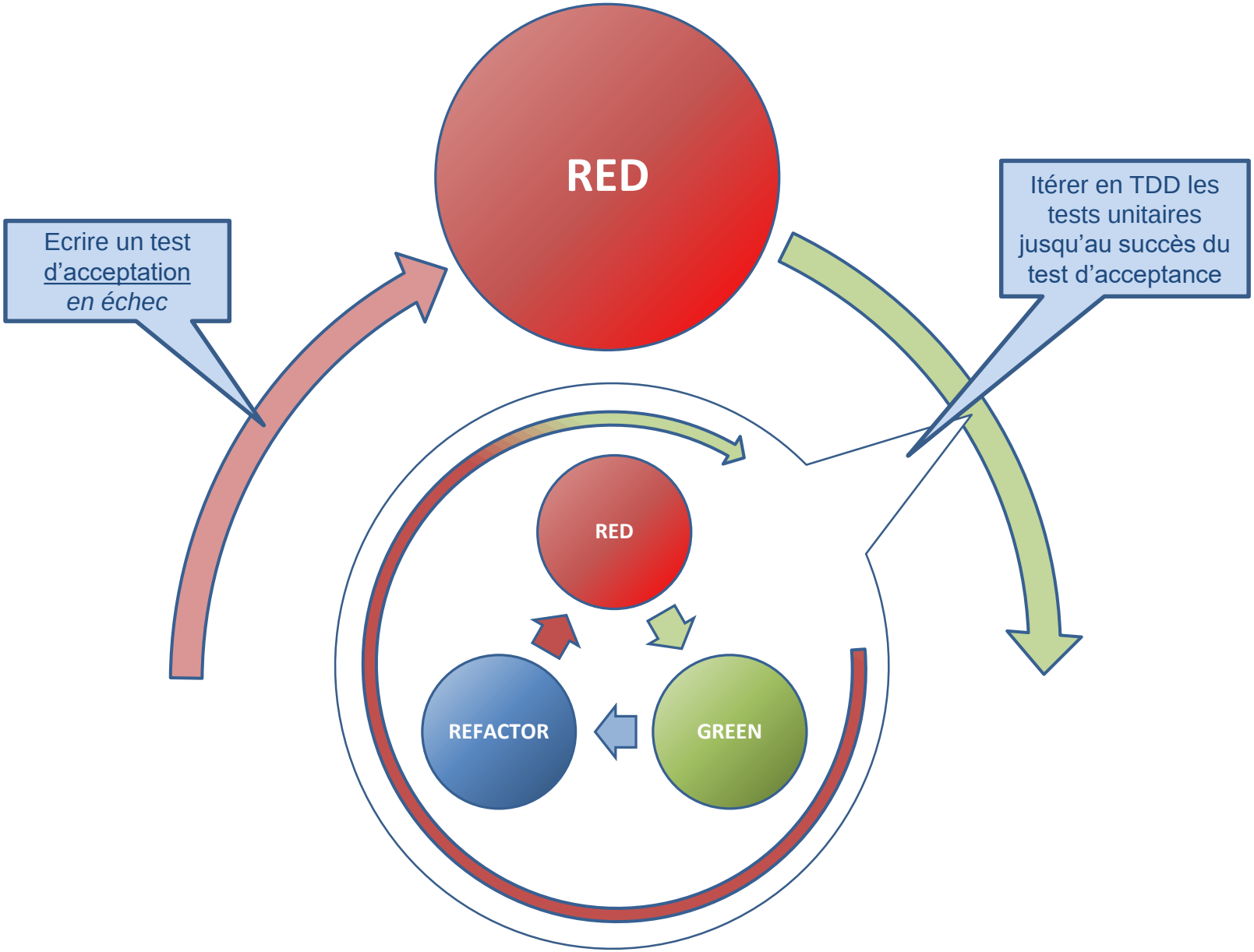
Le TDD « Outside In »

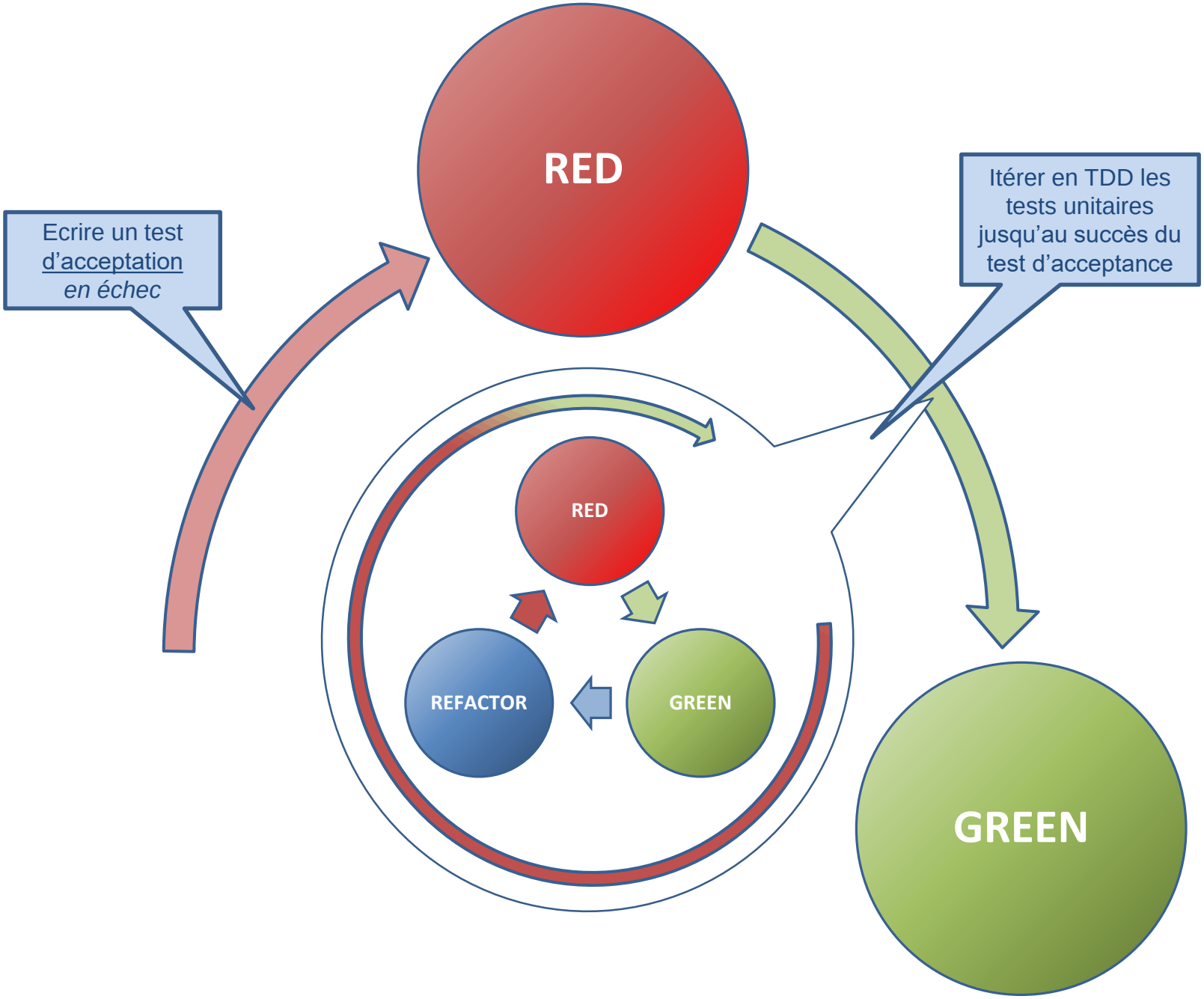

RED
Ecrire un test
d'acceptation
en échec

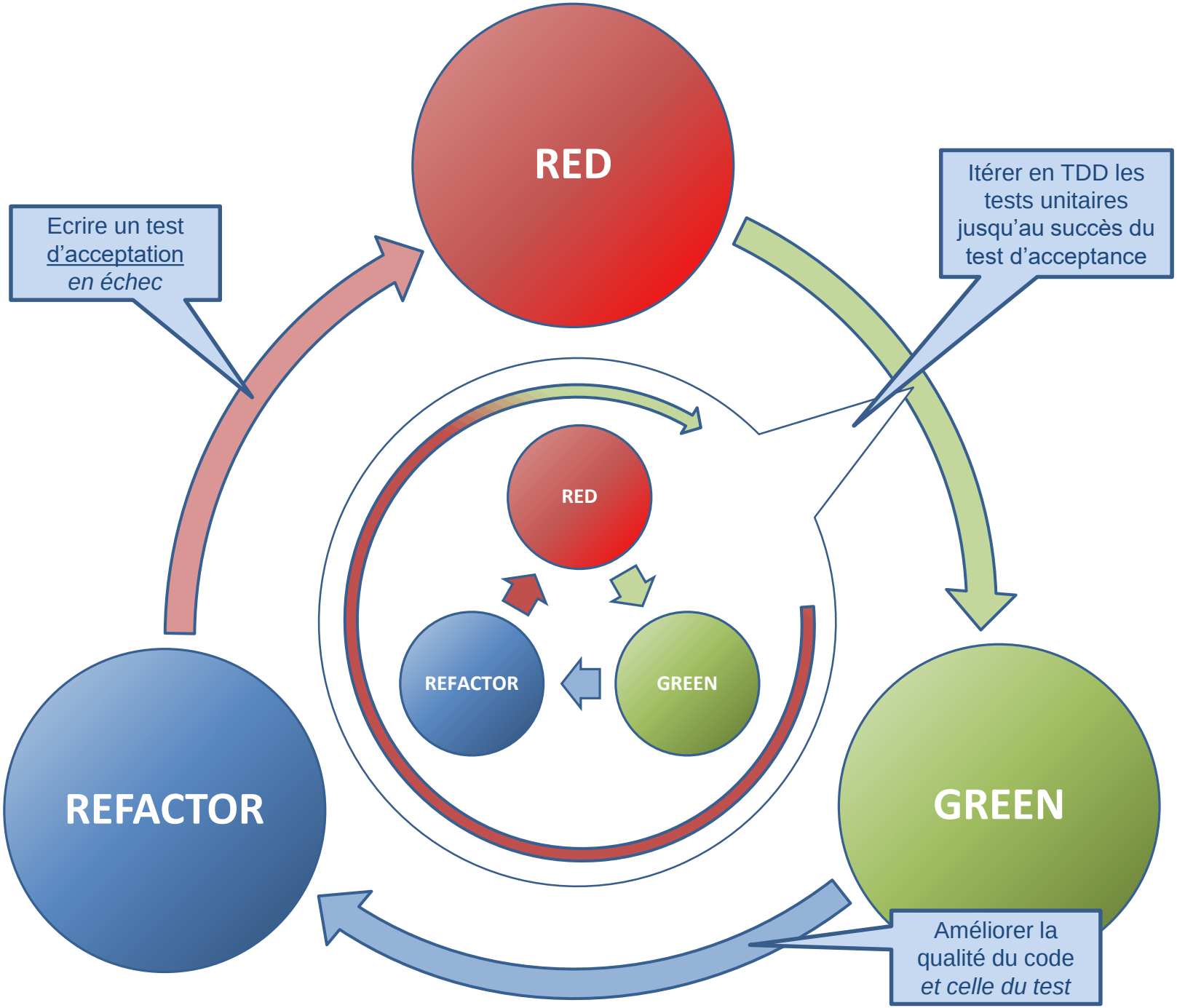

RED
Ecrire un test
d'acceptation
en échec
Ecrire un test
unitaire
en échec
RED

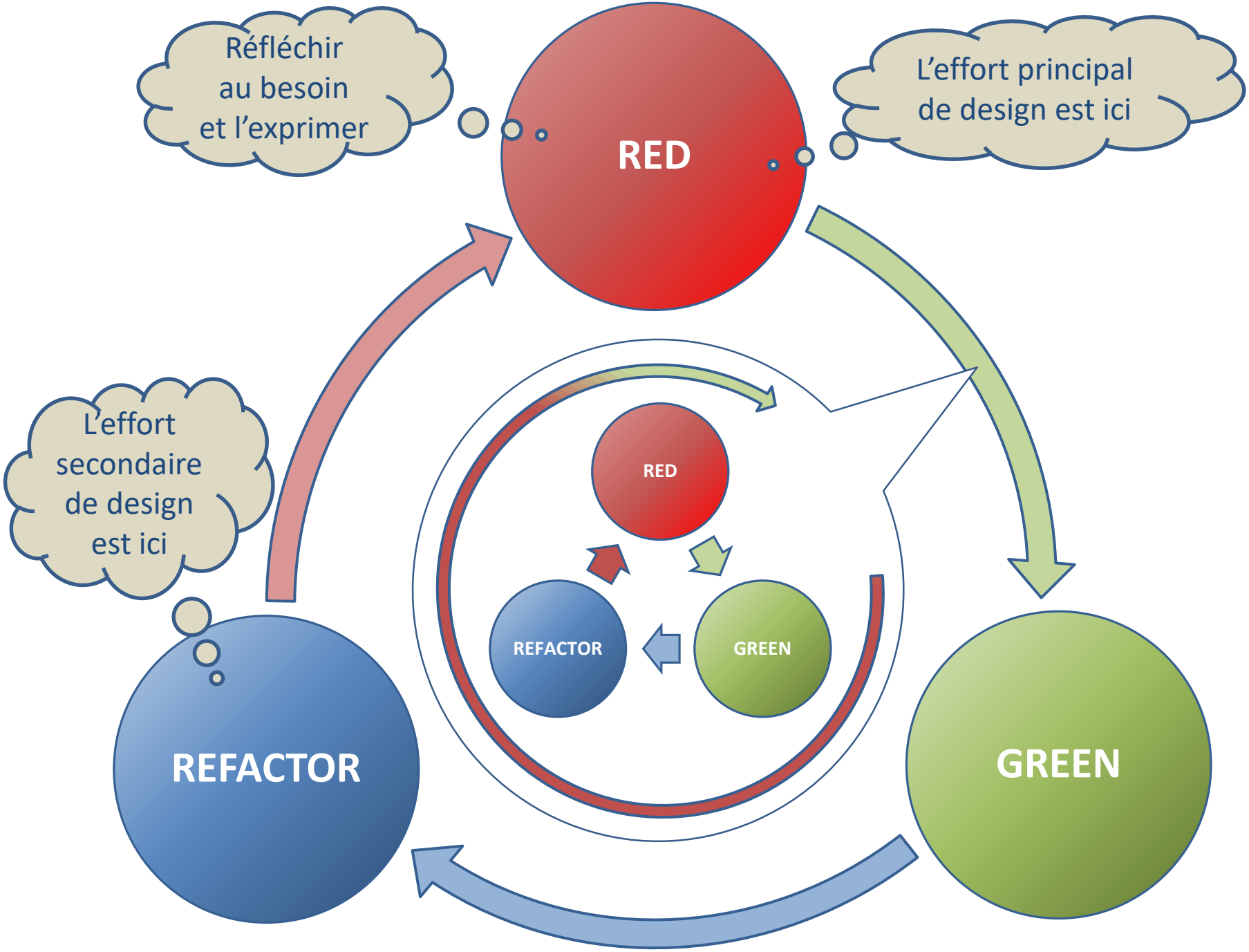

RED
Ecrire un test
d'acceptation
en échec
RED
Réaliser le code
répondant au test
et à rien d'autre
GREEN

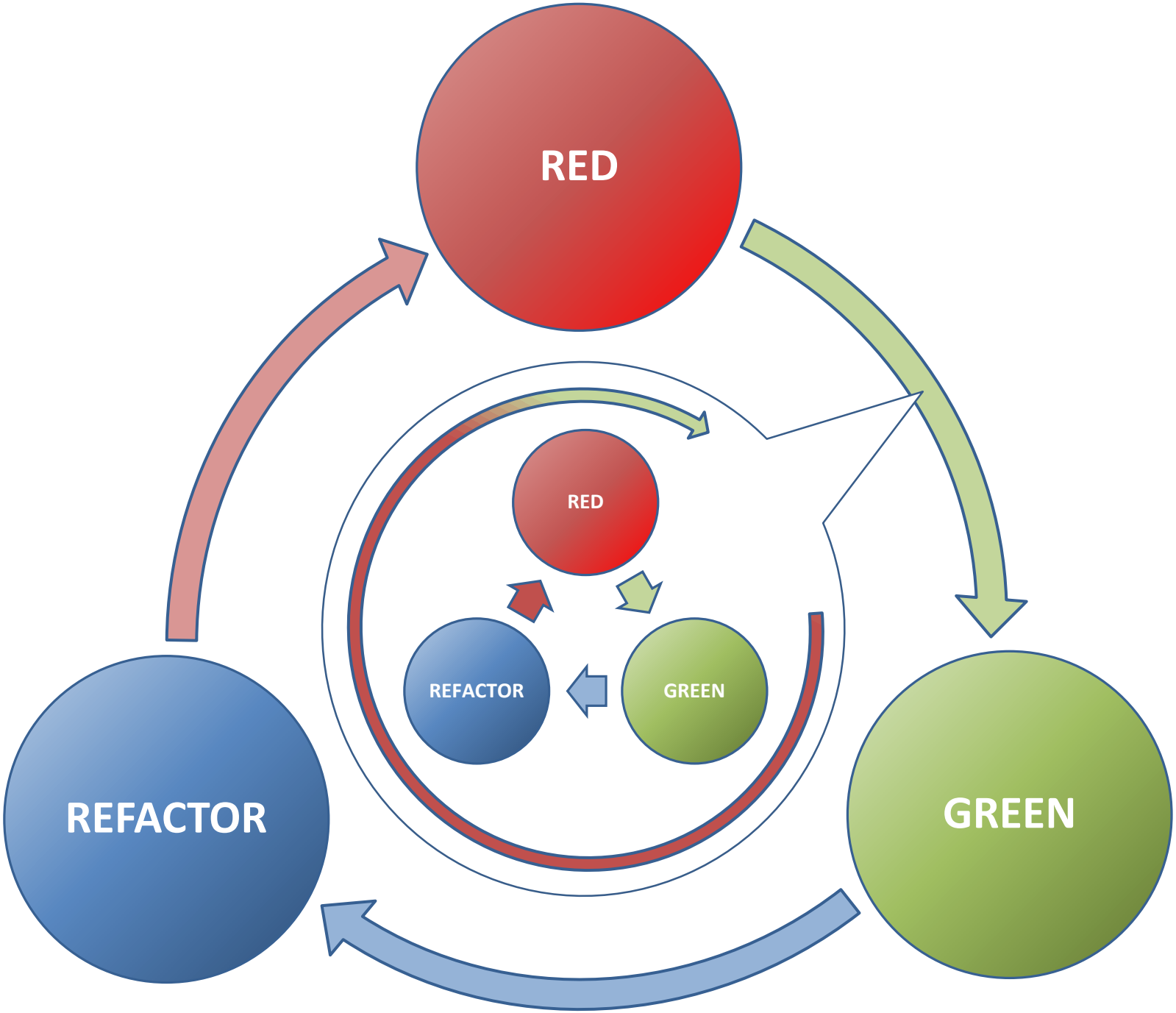

RED
Ecrire un test
d'acceptation
en échec
RED
REFACTOR
GREEN
Améliorer la
qualité du code
et celle du test


RED
Ecrire un test d'acceptation en échec
Itérer en TDD les tests unitaires jusqu'au succès du test d'acceptance
RED
REFACTOR
GREEN


RED
Ecrire un test d'acceptation en échec
Itérer en TDD les tests unitaires jusqu'au succès du test d'acceptance
RED
REFACTOR
GREEN
GREEN

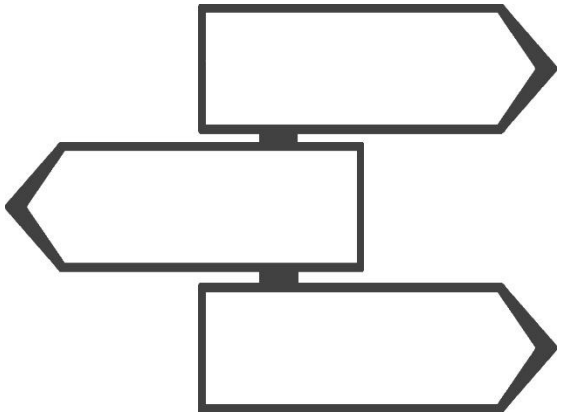

RED
Ecrire un test
d'acceptation
en échec
Itérer en TDD les
tests unitaires
jusqu'au succès du
test d'acceptance
RED
REFACTOR
GREEN
REFACTOR
GREEN
Améliorer la
qualité du code
et celle du test


Réfléchir
au besoin
et l'exprimer
RED
L'effort principal
de design est ici
L'effort
secondaire
de design
est ici
REFACTOR
GREEN
RED
REFACTOR
GREEN

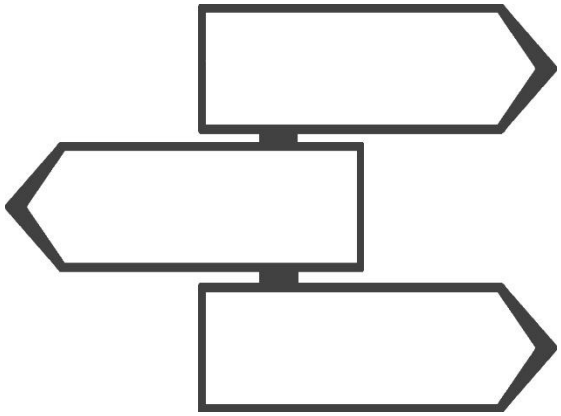

RED
GREEN
REFACTOR
RED
GREEN
REFACTOR

Oui mais ...

... dans quel but ?



Le cas d'utilisation comme axe central



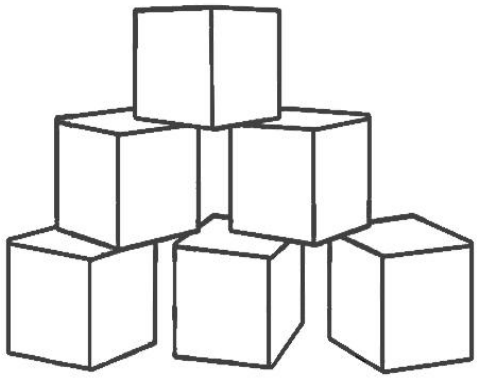
Le cas d'utilisation comme axe central

Des cas d'utilisation « encadrants » plus proches du métier

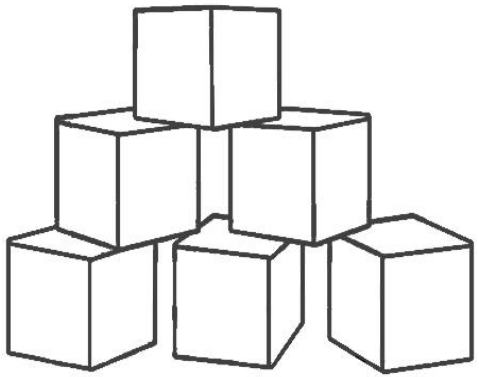
Un indicateur d'avancement dans le produit, sur la durée

L'effort de conception survient plus tôt dans le cycle

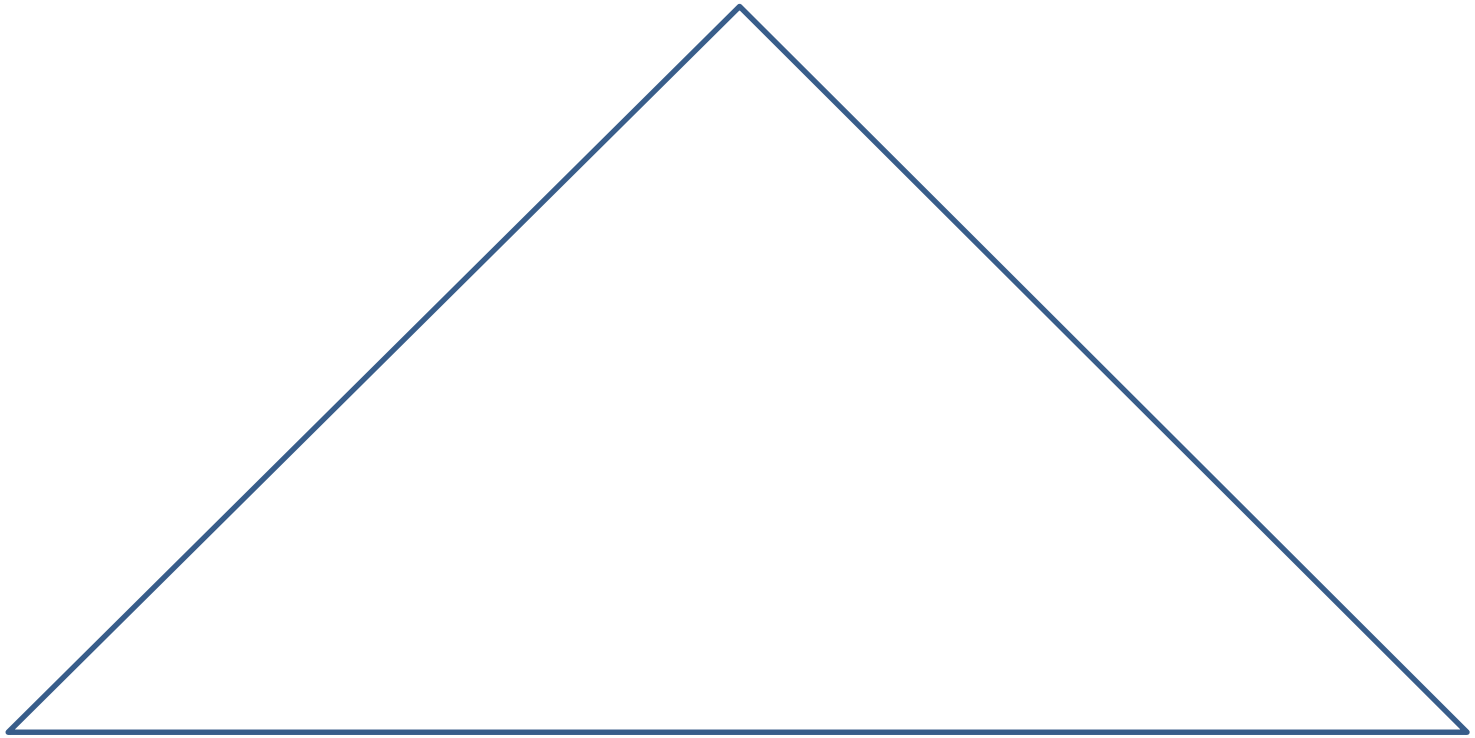
La proportion de tests réellement unitaires est facilitée

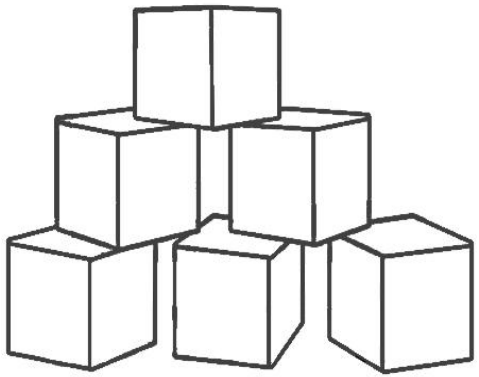


La pyramide des tests

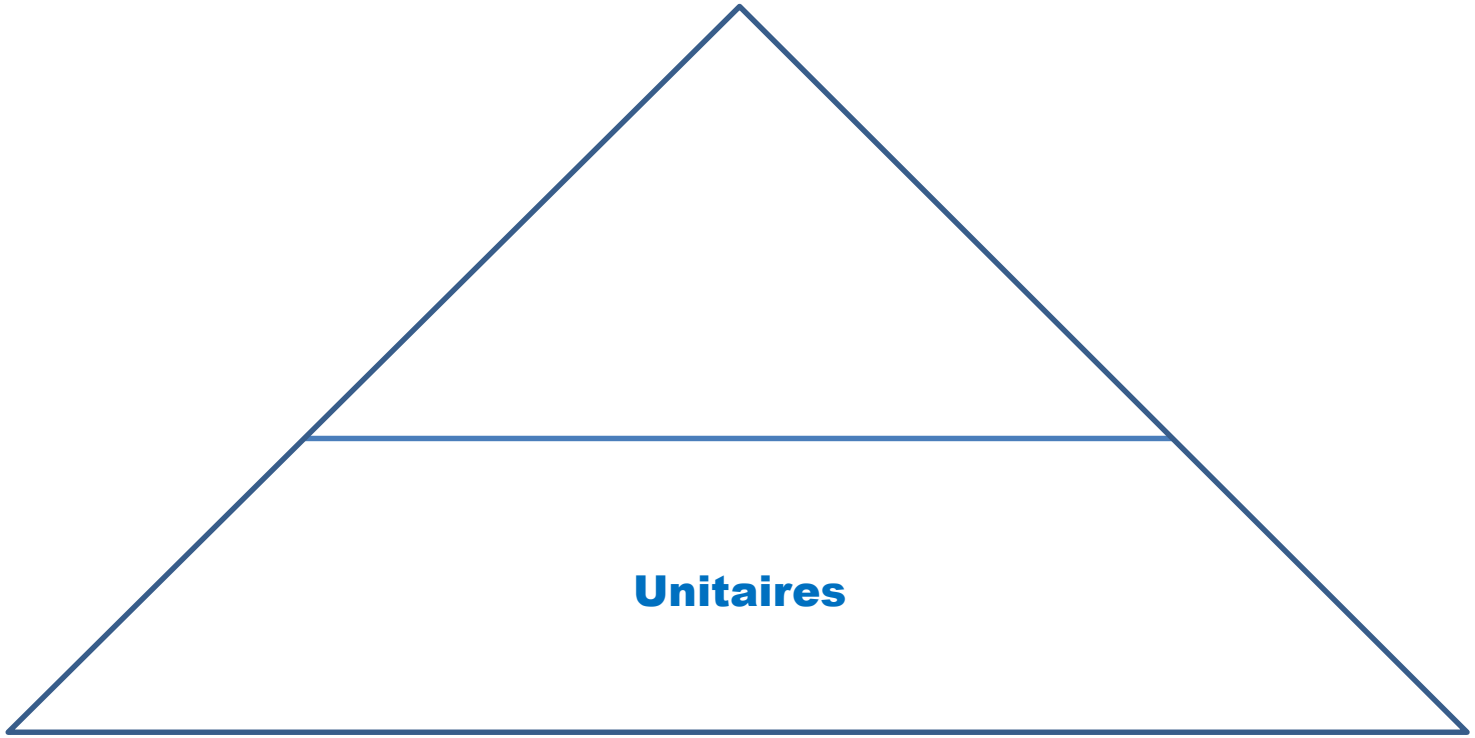


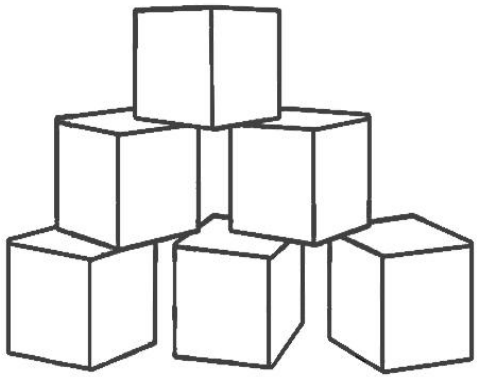
La pyramide des tests



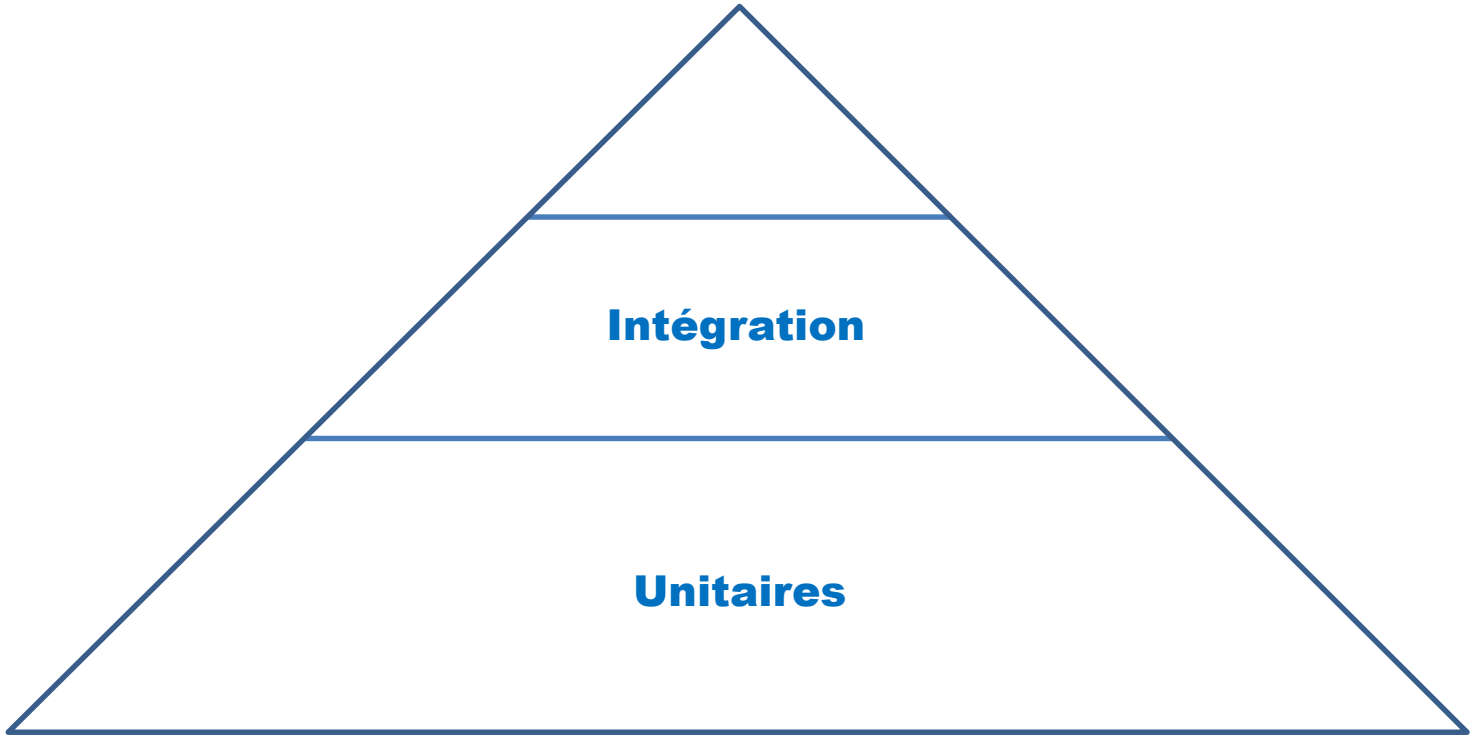


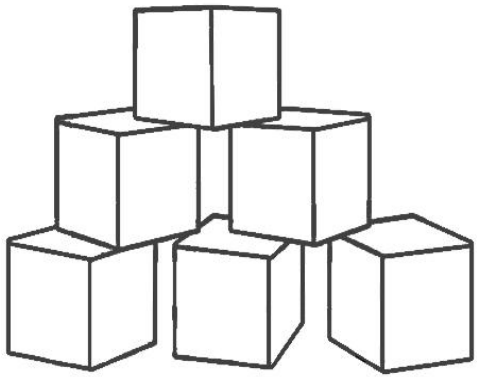
La pyramide des tests



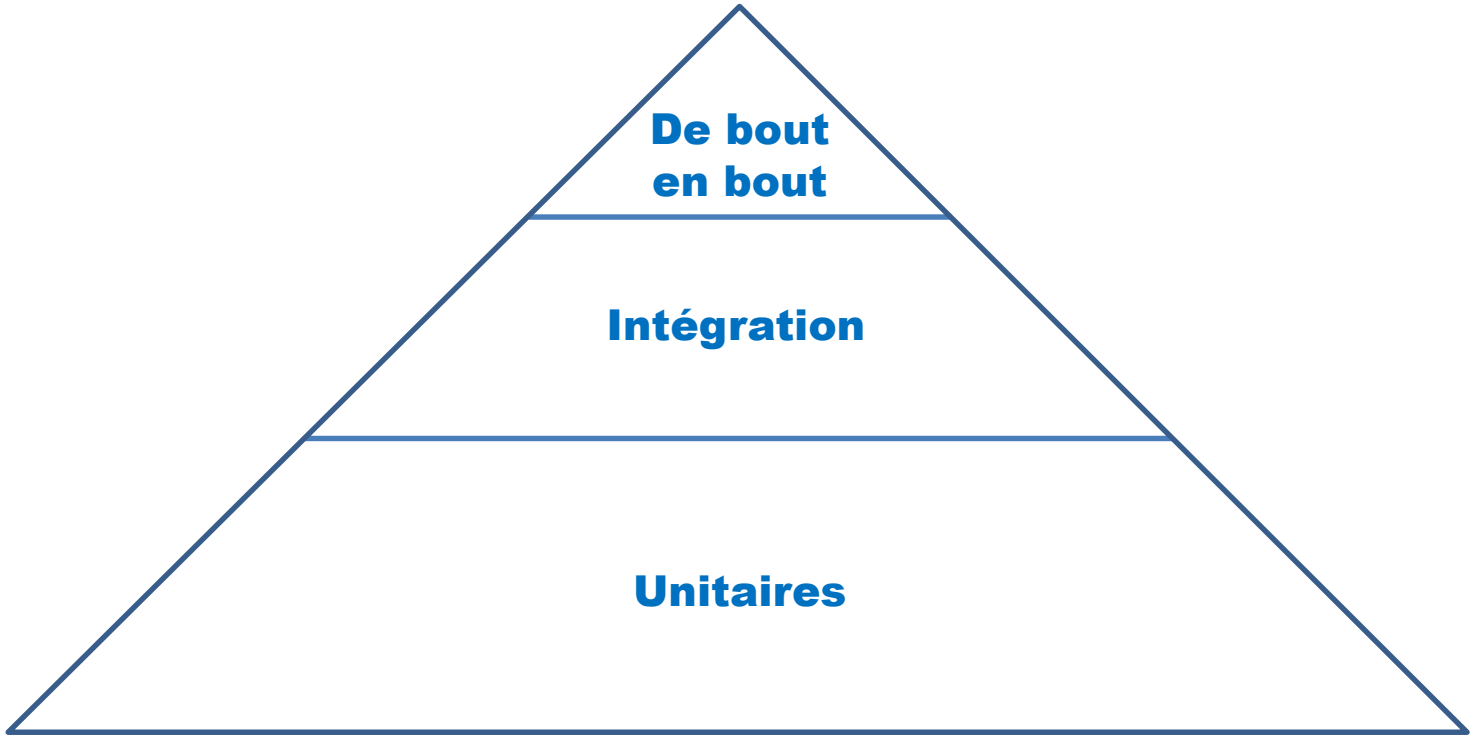


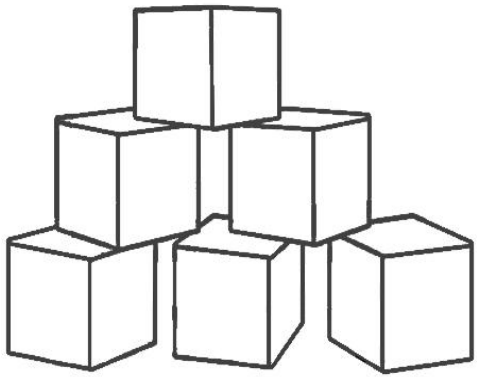
La pyramide des tests



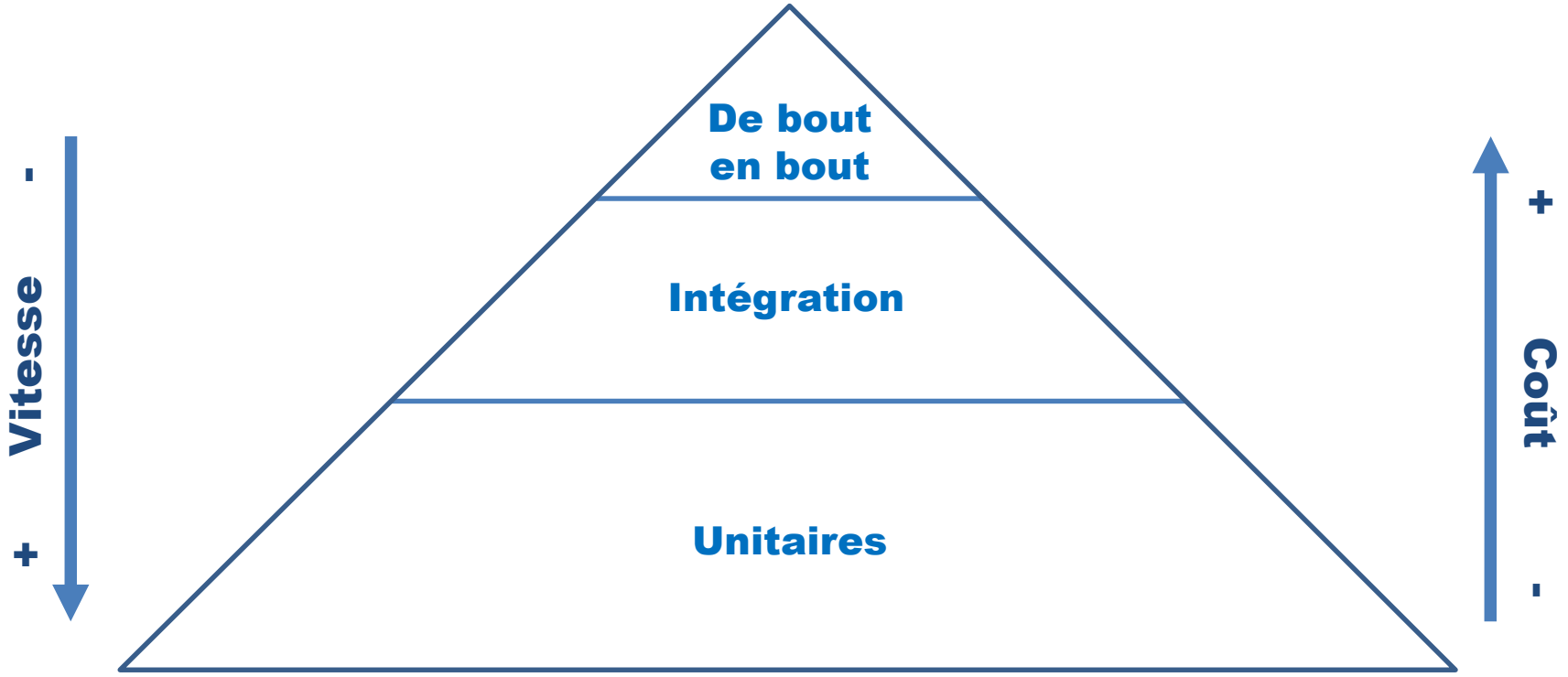


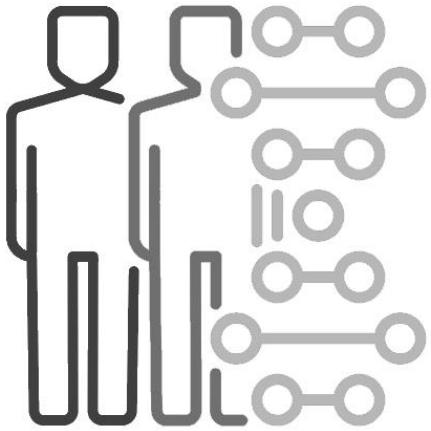
La pyramide des tests



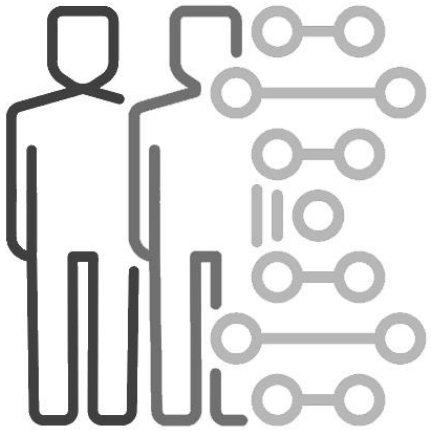


La pyramide des tests





Un allié de poids : les doublures de test



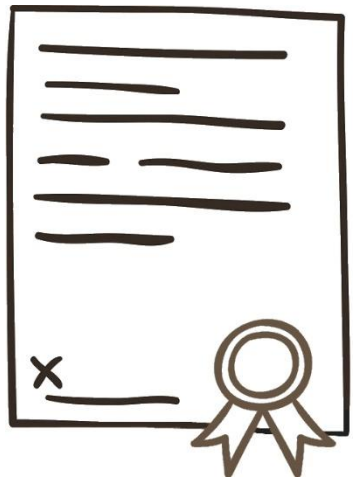
Un allié de poids : les doublures de test

Dummy, Fake, Stub, Spy, Mock … une simplification : Mock

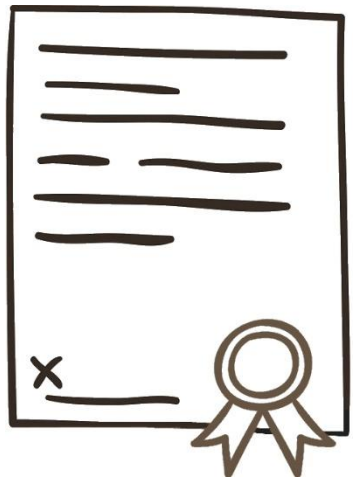
Des collaborateurs n'exposant que leurs interfaces

Leurs comportements et interactions sont vérifiés, pas leurs états

Un test n'a réellement qu'une seule raison d'échouer



L'interface publique en tant que contrat



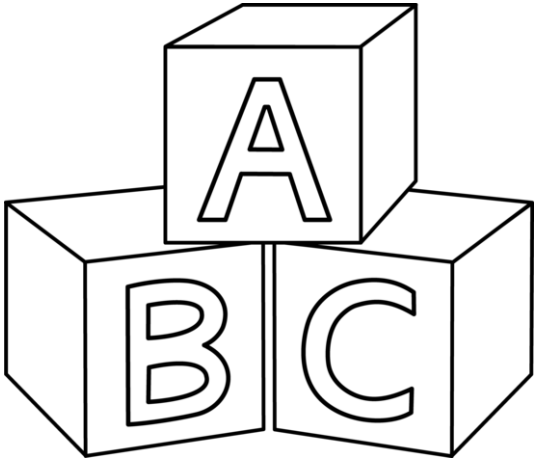
L'interface publique en tant que contrat

Les interfaces des objets métier définissent leur comportement

La séparation des responsabilités et dépendances est facilitée

Les tests unitaires peuvent pleinement l'être, car en isolation

Quelques rappels



4 règles élémentaires de conception

Tous les tests sont au vert

Le code révèle l'intention

Pas de duplication

Pas d'élément superflu

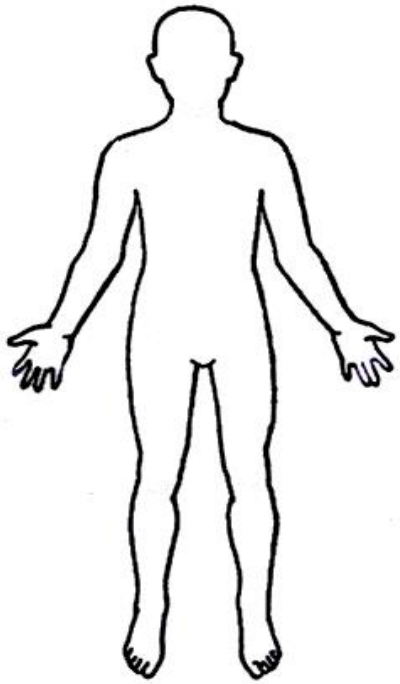
Quelques guidelines pour le cycle TDD

Ajouter un (et un seul) nouveau test seulement « au vert »
(à un même niveau de test : unitaire / acceptation)

Voir échouer le test avant de coder sa solution

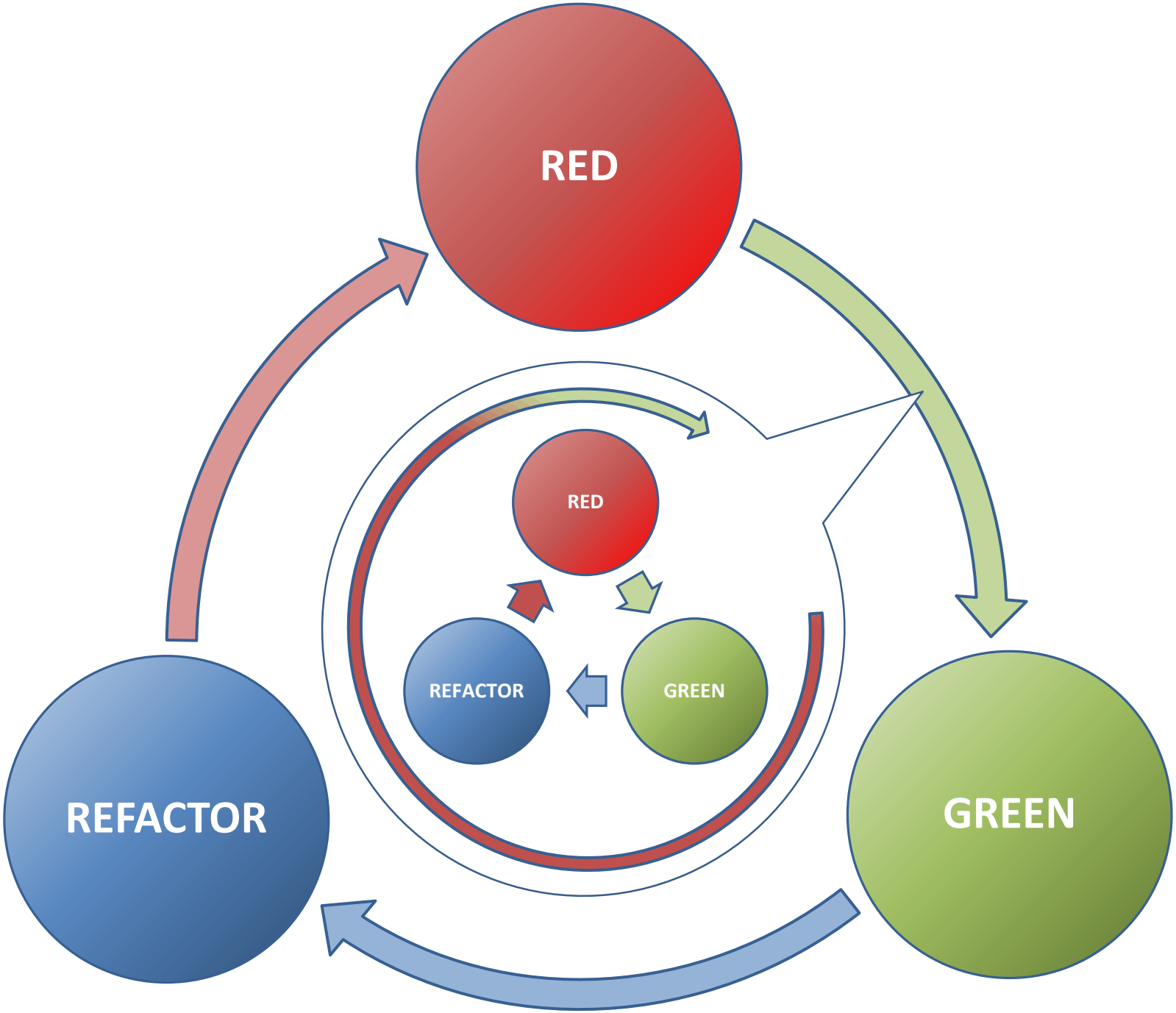
Coder dans l'optique de revenir au plus vite « au vert »

Effectuer un refactoring du code ou du test, pas des deux à la fois

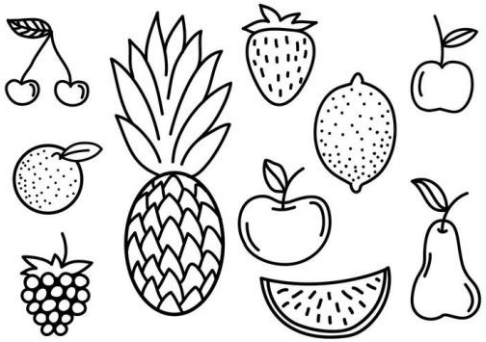


Anatomie d'un test

3	GIVEN	Contexte	= états / données
2	WHEN	Événement	= ce qui est testé
1	THEN	Attendu	= réponse au besoin


RED
GREEN
REFACTOR
RED
GREEN
REFACTOR

Codons un peu !



Fruit Shop

Faire payer le bon montant quand le client passe en caisse.

Règles de gestion (prix en centimes) :

- **Une pomme coûte 100**
- **Une banane coûte 150**
- **Une cerise coûte 75**

FruitShopTest

```
@Test
public void neRienEncaisserPourUnPanierVide() {

}
```

FruitShopTest

```
@Test
public void neRienEncaisserPourUnPanierVide() {

    // THEN
    assertThat(montantTotal).isEqualTo(0);
}
```

FruitShopTest

```
@Test
public void neRienEncaisserPourUnPanierVide() {

    // WHEN
    int montantTotal = encaissement.calculerMontantTotal();

    // THEN
    assertThat(montantTotal).isEqualTo(0);
}
```

FruitShopTest

```
private Encaissement encaissement = new Encaissement();

@Test
public void neRienEncaisserPourUnPanierVide() {

    // WHEN
    int montantTotal = encaissement.calculerMontantTotal();

    // THEN
    assertThat(montantTotal).isEqualTo(0);
}
```

FruitShopTest

```
private Encaissement encaissement = new Encaissement(panier);

@Test
public void neRienEncaisserPourUnPanierVide() {

    // GIVEN
    // Le panier est vide

    // WHEN
    int montantTotal = encaissement.calculerMontantTotal();

    // THEN
    assertThat(montantTotal).isEqualTo(0);
}
```

FruitShopTest

```
private Panier panier = new Panier();
private Encaissement encaissement = new Encaissement(panier);

@Test
public void neRienEncaisserPourUnPanierVide() {

    // GIVEN
    // Le panier est vide

    // WHEN
    int montantTotal = encaissement.calculerMontantTotal();

    // THEN
    assertThat(montantTotal).isEqualTo(0);
}
```

EncaissementTest

```
@Test
public void neRienEncaisserPourUnPanierVide() {
}
```

EncaissementTest

```
@Test
public void neRienEncaisserPourUnPanierVide() {

    // THEN
    assertThat(montantTotal).isEqualTo(0);
}
```

EncaissementTest

```
@Test
public void neRienEncaisserPourUnPanierVide() {

    // WHEN
    int montantTotal = encaissement.calculerMontantTotal();

    // THEN
    assertThat(montantTotal).isEqualTo(0);
}
```

EncaissementTest

```
private Encaissement encaissement = new Encaissement();

@Test
public void neRienEncaisserPourUnPanierVide() {

    // WHEN
    int montantTotal = encaissement.calculerMontantTotal();

    // THEN
    assertThat(montantTotal).isEqualTo(0);
}
```

EncaissementTest

```
private Encaissement encaissement = new Encaissement(panier);

@Test
public void neRienEncaisserPourUnPanierVide() {

    // GIVEN
    given(panier.listerFruits())
        .willReturn(Collections.emptyList());

    // WHEN
    int montantTotal = encaissement.calculerMontantTotal();

    // THEN
    assertThat(montantTotal).isEqualTo(0);
}
```

EncaissementTest

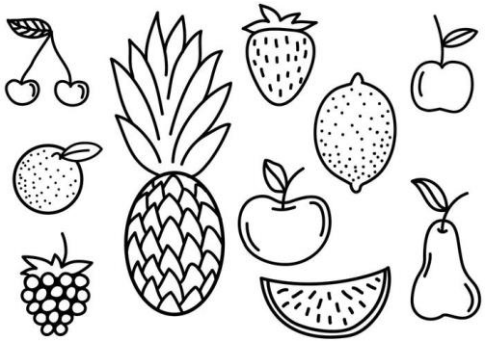
```
private Panier panier = new Mock(Panier.class);
private Encaissement encaissement = new Encaissement(panier);

@Test
public void neRienEncaisserPourUnPanierVide() {

    // GIVEN
    given(panier.listerFruits())
        .willReturn(Collections.emptyList());

    // WHEN
    int montantTotal = encaissement.calculerMontantTotal();

    // THEN
    assertThat(montantTotal).isEqualTo(0);
}
```

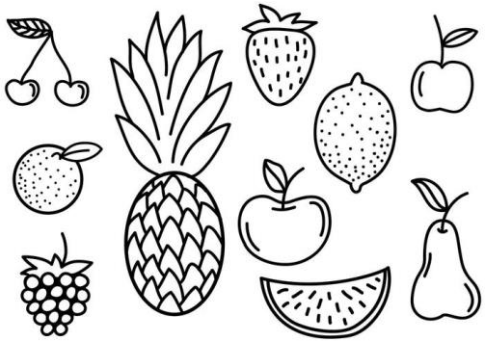


Fruit Shop

Faire payer le bon montant quand le client passe en caisse.

Règles de gestion (prix en centimes) :

- **Une pomme coûte 100**
- **Une banane coûte 150**
- **Une cerise coûte 75**

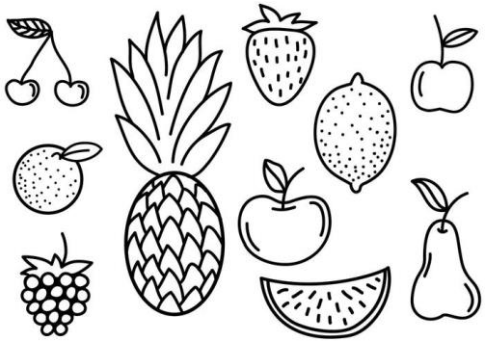


Fruit Shop

Faire payer le bon montant quand le client passe en caisse.

Règles de gestion (prix en centimes) :

- **Une pomme coûte 100**
- **Une pomme offerte pour deux pommes achetées**
- **Une banane coûte 150**
- **Une cerise coûte 75**

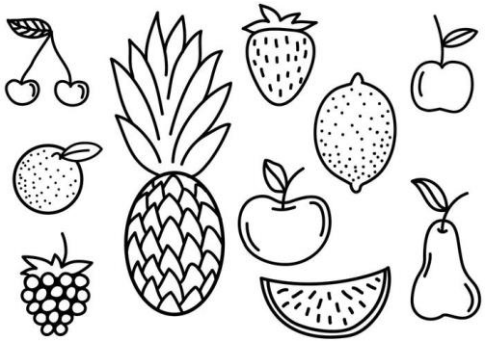


Fruit Shop

Faire payer le bon montant quand le client passe en caisse.

Règles de gestion (prix en centimes) :

- **Une pomme coûte 100**
- **Une pomme offerte pour deux pommes achetées**
- **Une banane coûte 150**
- **La deuxième banane est à moitié prix**
- **Une cerise coûte 75**



Fruit Shop

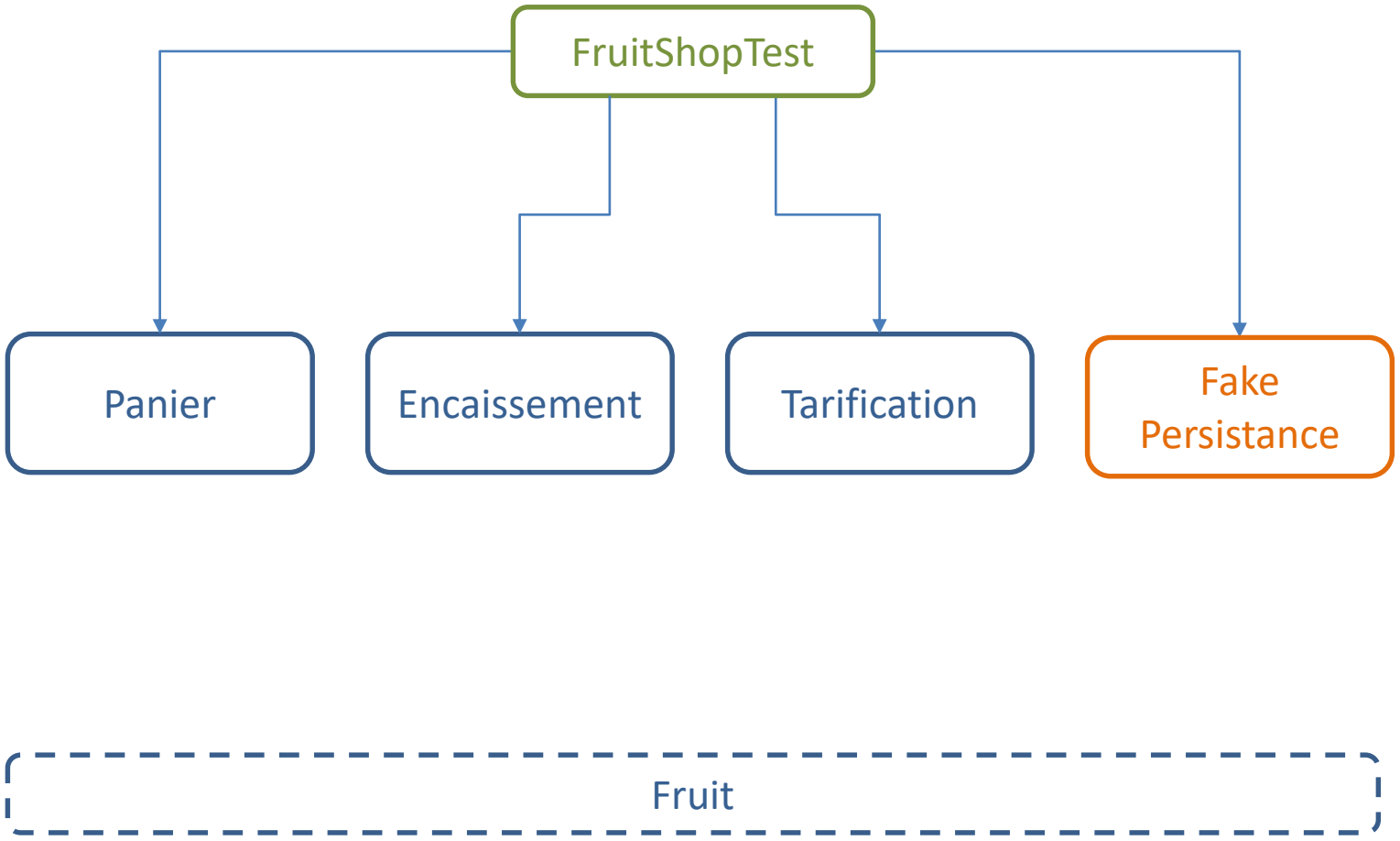
Faire payer le bon montant quand le client passe en caisse.

Règles de gestion (prix en centimes) :

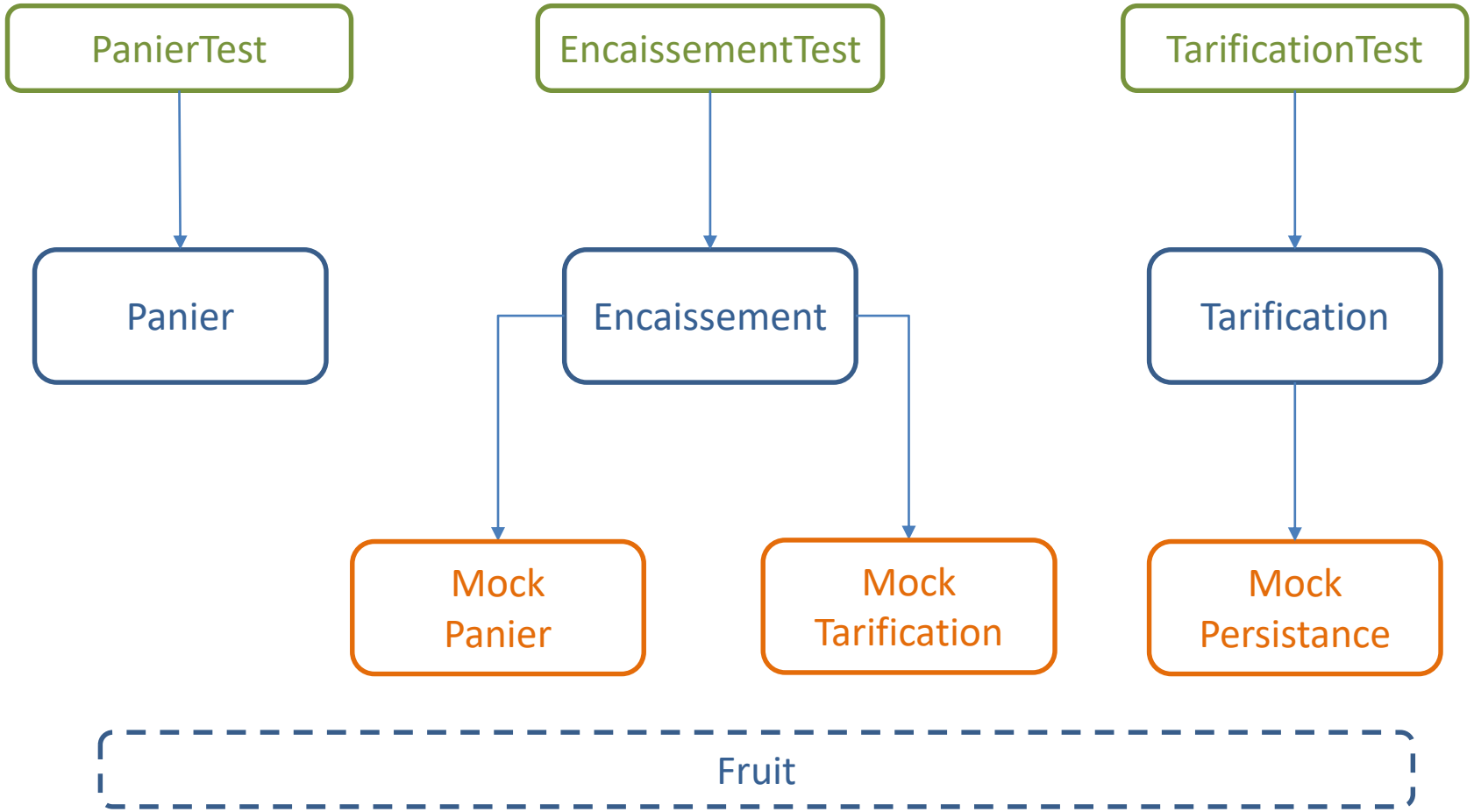
- **Une pomme coûte 100**
- **Une pomme offerte pour deux pommes achetées**
- **Une banane coûte 150**
- **La deuxième banane est à moitié prix**
- **Une cerise coûte 75**
- **Un client fidèle a droit à 10% de remise**

Une solution possible

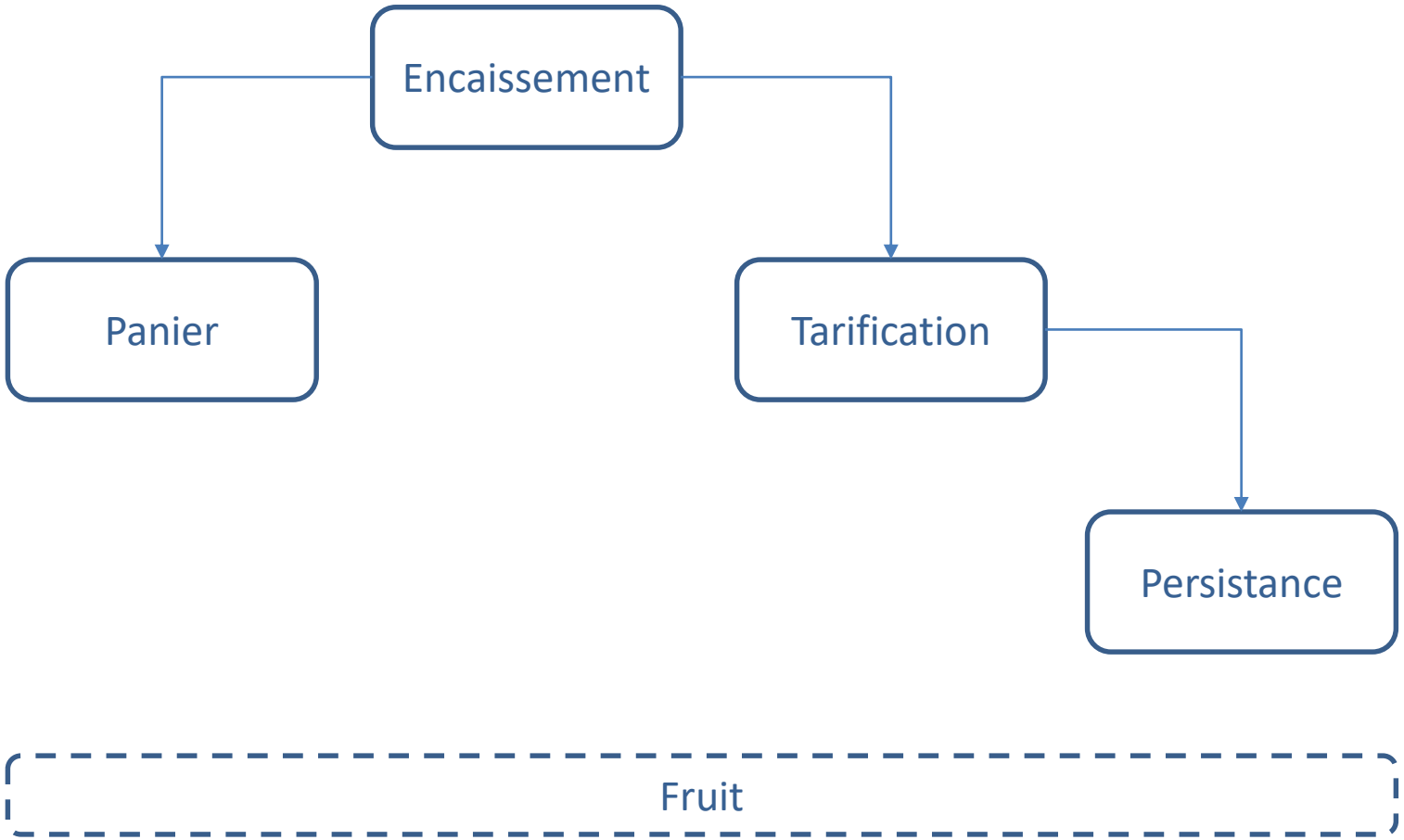
Test d'acceptation

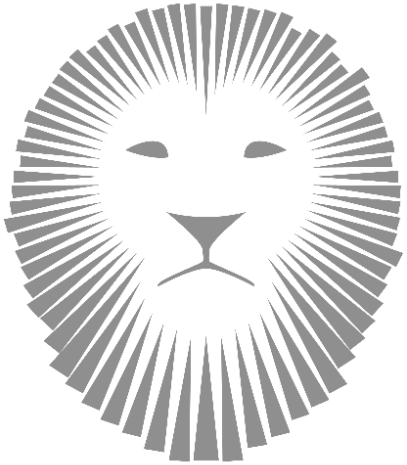


Tests unitaires



Implémentation





Merci à tous !

lyontechhub.slack.com