

Tests

ESIR2 SNR - GLA



Tests


"Le test est un processus manuel ou automatique, qui vise à établir qu'un système vérifie les propriétés exigées par sa spécification, ou à détecter des différences entre les résultats engendrés par le système et ceux qui sont attendus par la spécification "

Extrait de la norme IEEE-STD729, 1983.

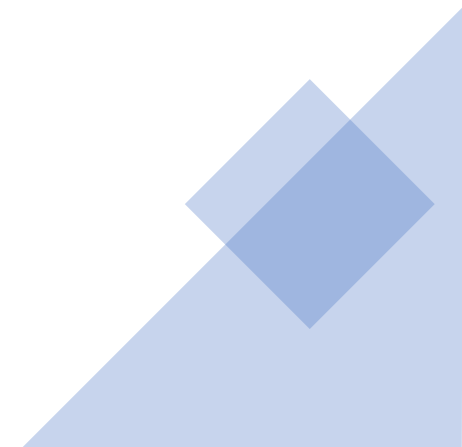
Est-ce que ça marche ?

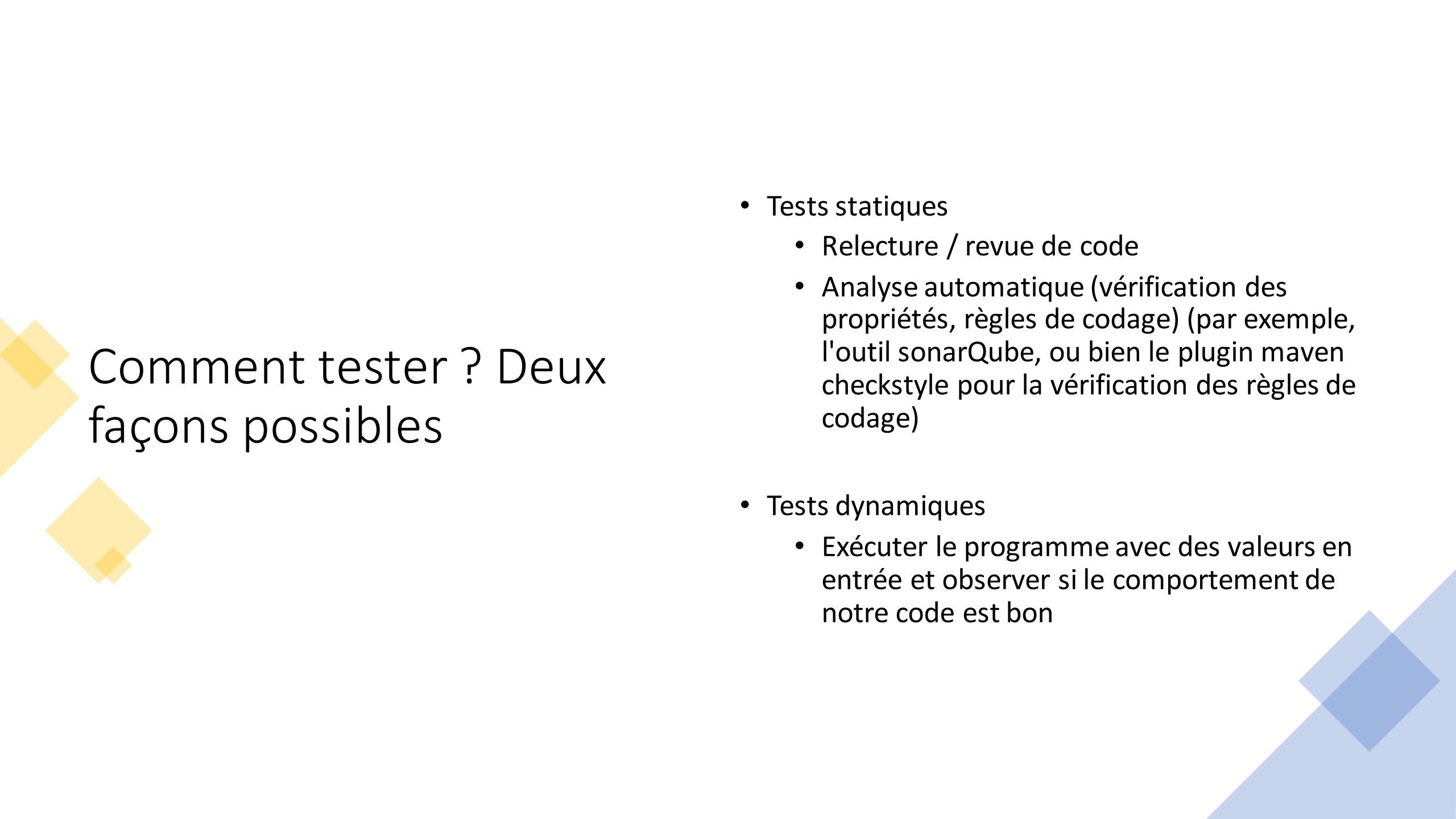
Est-ce que ça marche comme prévu ?





Tester c'est bien, mais
que tester ?

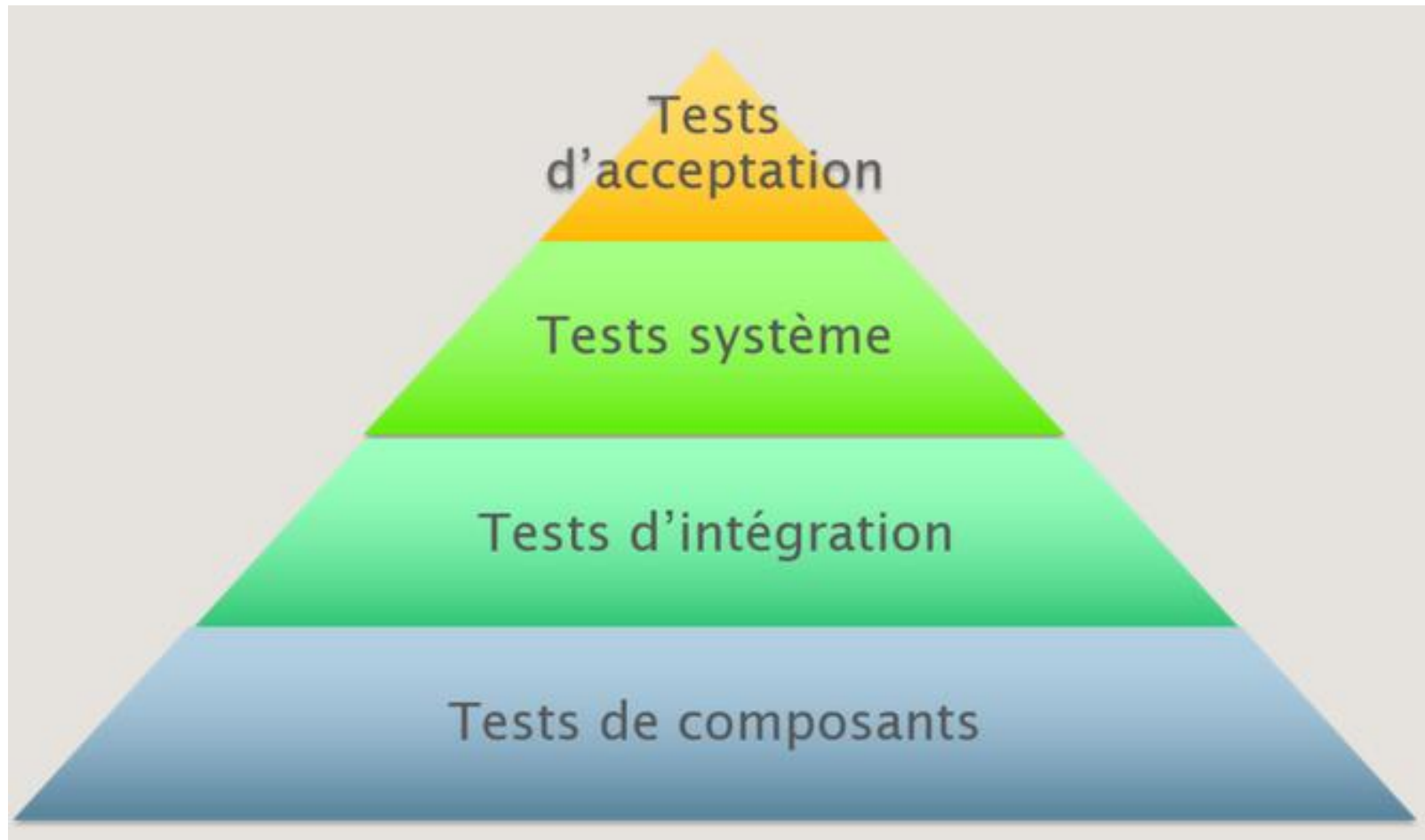
- Fonctionnalité
 - Sécurité / intégrité
 - Utilisabilité
 - Cohérence
 - Maintenabilité
 - Efficacité
 - Robustesse
 - Sûreté de fonctionnement
 - Etc.
- 

The slide features decorative geometric shapes in the corners. On the left, there are several overlapping yellow squares and diamonds of various sizes. On the bottom right, there are overlapping blue squares and diamonds. The main title is positioned to the right of the yellow shapes on the left side of the slide.

Comment tester ? Deux façons possibles

- Tests statiques
 - Relecture / revue de code
 - Analyse automatique (vérification des propriétés, règles de codage) (par exemple, l'outil sonarQube, ou bien le plugin maven checkstyle pour la vérification des règles de codage)
- Tests dynamiques
 - Exécuter le programme avec des valeurs en entrée et observer si le comportement de notre code est bon

Types de tests

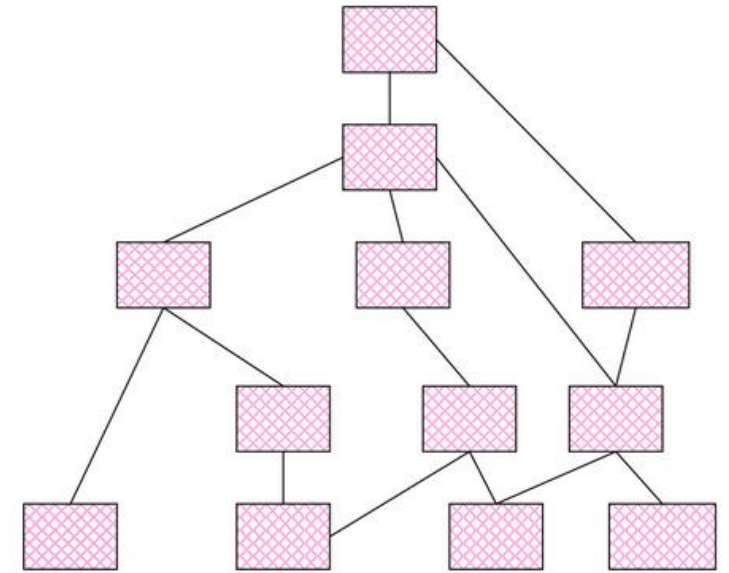


Tests unitaires

- tester les différents composants du logiciel séparément afin de s'assurer que chaque élément fonctionne comme spécifié
- Ces tests sont aussi appelés test unitaires et sont généralement écrits et exécutés par le développeur qui a écrit le code du composant.
- Pour une authentification, le bouton « se connecter » est un composant.
- Ces tests sont (ou doivent théoriquement) toujours automatisés.

Tests d'intégration

- tests effectués entre les composants afin de s'assurer du fonctionnement des interactions et de l'interface entre les différents composants. Ces tests incluent l'ordre d'intégration des différents modules en fonction de leur couplage
- Ces tests sont également gérés, en général, par des développeurs.
- Toujours depuis l'authentification ici on vérifie que le message envoyé après l'appui sur le bouton « se connecter » est bien reçu par le serveur d'authentification.
- Ces tests peuvent être manuels ou automatisés.



Tests système

- c'est généralement les seuls qui sont effectués par les ingénieurs de tests.
- Leurs but est de vérifier que le système (le logiciel ou l'application dans son ensemble) répond aux exigences définies dans les spécifications.
- Ici on vérifie que l'authentification fonctionne bien, que les bonnes erreurs sont remontées...
- Ces tests peuvent être manuels ou automatisés,

Tests d'acceptation (de recette)

- Les tests « finaux » effectués par le métier ou les utilisateurs finaux (par exemple avec une bêta test). Leur but est de confirmer que le produit final correspond bien aux besoins des utilisateurs finaux.
- Attention : ce n'est pas parce qu'une application répond aux spécifications qu'elle répond aux besoins des utilisateurs. Cela peut arriver pour plusieurs raisons telles que des problèmes (ou trous) dans les spécifications, des problèmes d'ergonomie...
- Avec ces tests on vérifie qu'en plus de répondre aux exigences l'authentification correspond bien à ce à quoi le métier ou les clients finaux s'attendent (un champ authentification trop petit peut être problématique par exemple).
- Ces tests sont manuels

Tests de non-régression

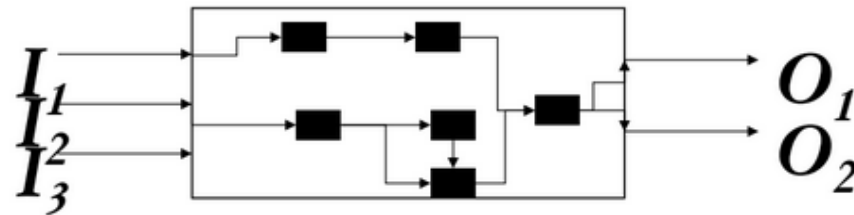
- Des tests réalisés lors de la phase de maintenance
 - Après un refactoring (ajout/suppression de fonctionnalités)
- L'objectif est de déterminer si les modifications apportées au logiciel n'ont pas introduit de nouvelles erreurs
 - Vérifier que ce qui marchait marche encore

Tests structurels et fonctionnels

- Tests fonctionnels : tests de la boîte noire
 - Utilise la description des fonctionnalités du programme



- Tests structurels : tests de la boîte blanche
 - Utilise la structure interne du programme



Tests structurels et fonctionnels

factoriel(x) est une fonction qui retourne le factoriel d'un entier x pris en entrée.

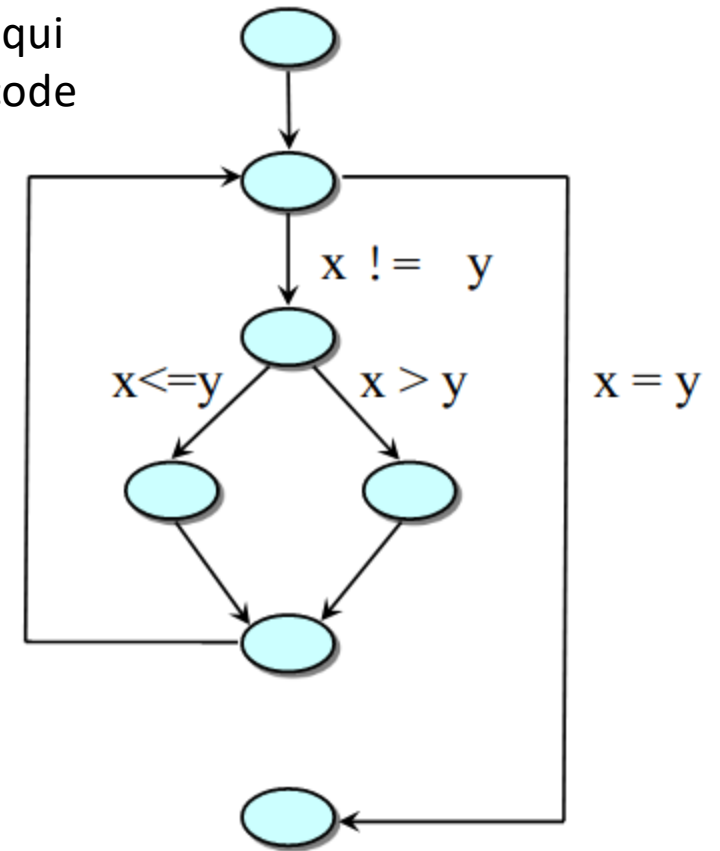
Sans se soucier de la façon dont cette fonction est implémentée, un test fonctionnel consiste à choisir des valeurs en entrée, et les comparer avec les résultats souhaités en sortie. Ce type de tests est indispensable surtout pour les limites.

Dans notre cas, une limite serait le chiffre 0 (ou 1). Le test serait utile pour déterminer si :

Factoriel(0) = 1

Graphe de flot de contrôle qui représente la structure du code

```
read(x);  
read(y);  
while x != y loop  
  if x > y then  
    x := x - y;  
  else  
    y := y - x;  
  end if;  
end loop;  
gcd := x;
```



Pour chaque route du graphe, un ou plusieurs tests structurels devront être écrits afin de s'assurer du bon comportement de la route

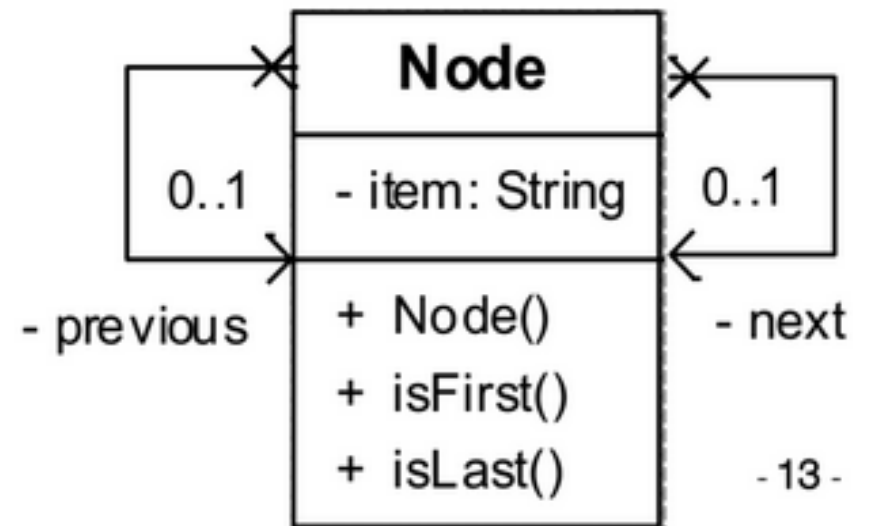
Tests structurels et fonctionnels

- Les critères structurels et fonctionnels sont complémentaires
 - Une erreur d'omission ne peut être détectée par le test structurel
 - Du code mort ne peut pas être détecté par le test fonctionnel
- Au niveau unitaire
 - On commence par le test fonctionnel
 - On complète par du test structurel

Tests unitaire

- Pour un langage procédural
 - Unité de test = procédure
- Dans un contexte orienté objet
 - Unité de test = classe

```
void Ouvrir (char *nom, Compte *C, float S, float D )  
{  
    C->titulaire = AlloueEtCopieNomTitulaire(nom);  
    (*C).montant = S ;  
    (*C).seuil = D ;  
    (*C).etat = DEJA_OUVERT ;  
    (*C).histoire.nbop = 0;  
    EnregistrerOperation(C);  
    EcrireTexte("Ouverture du compte numero ");  
    EcrireEntier(NumeroCourant+1);  
    EcrireTexte(", titulaire : ");  
    EcrireTexte(C->titulaire); EcrireCar("");  
    ALaLigne();  
}
```



Tests unitaires en python

- Se font à l'aide du module `unittest` depuis la librairie standard de python
- Pour écrire des tests, il faut :
 1. Importer le module **`unittest`**
 2. Créer une classe qui contient les différents tests, la classe hérite de **`unittest.TestCase`**
 3. Enfin, une méthode est requise pour chaque test. Le nom de chaque méthode doit débuter avec **`test_`** afin qu'elle soit considérée comme un test et exécutée avec le runner **`nose test`**. Le nom des méthodes de tests est généralement très long pour expliquer son objectif
 4. Avec cette structure mise en place, on peut enfin écrire des tests
 5. Rajouter la ligne suivante à la fin de votre fichier de tests.

```
if __name__ == '__main__':  
    unittest.main()
```

Ainsi, l'exécution des tests se fera avec la commande : **`$ python nomDuFichier.py`**

Tests unitaires en python

```
def addition(a,b):  
    resultat = a+b  
    print("a + b = " , resultat)  
    return resultat  
  
def multiplication(a,b):  
    resultat = a*b  
    print("a x b = ", resultat)  
    return resultat
```

Code.py

```
import unittest  
from code import addition,multiplication  
  
class Tests(unittest.TestCase):  
    def test_addition(self):  
        results = addition(1,2)  
        self.assertEqual(results, 3)  
  
    def test_multiplication(self):  
        results = multiplication(3, 4)  
        self.assertEqual (results, 13)  
  
if __name__ == '__main__':  
    unittest.main()
```

Test.py

Tests unitaires en python - exécution

```
echerfa@echerfa-Latitude-7490:~/Documents/python$ python3 tests.py
a + b = 3
.a x b = 12
F
=====
FAIL: test_multiplication (__main__.Tests)
-----
Traceback (most recent call last):
  File "/home/echerfa/Documents/python/tests.py", line 11, in test_multiplication
    self.assertEqual(results, 13)
AssertionError: 12 != 13
-----
Ran 2 tests in 0.000s

FAILED (failures=1)
echerfa@echerfa-Latitude-7490:~/Documents/python$
```

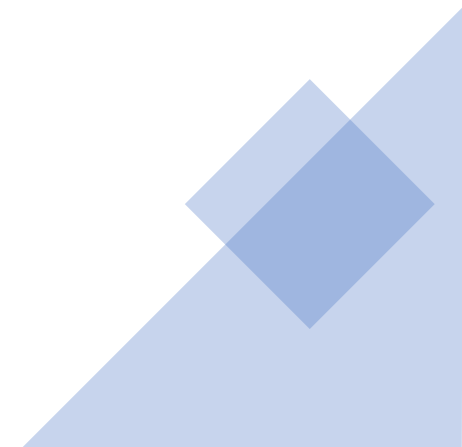
```
def test_test_If_Values_Are_Not_Numerical_For_Multiplication(self):
    with self.assertRaises(TypeError):
        multiplication('3','4')
```

Comment écrire et exécuter les tests sur python ?

- `assert`: base assert allowing you to write your own assertions
- `assertEqual(a, b)`: check a and b are equal
- `assertNotEqual(a, b)`: check a and b are not equal
- `assertIn(a, b)`: check that a is in the item b
- `assertNotIn(a, b)`: check that a is not in the item b
- `assertFalse(a)`: check that the value of a is False
- `assertTrue(a)`: check the value of a is True
- `assertIsInstance(a, TYPE)`: check that a is of type "TYPE"
- `assertRaises(ERROR, a, args)`: check that when a is called with args



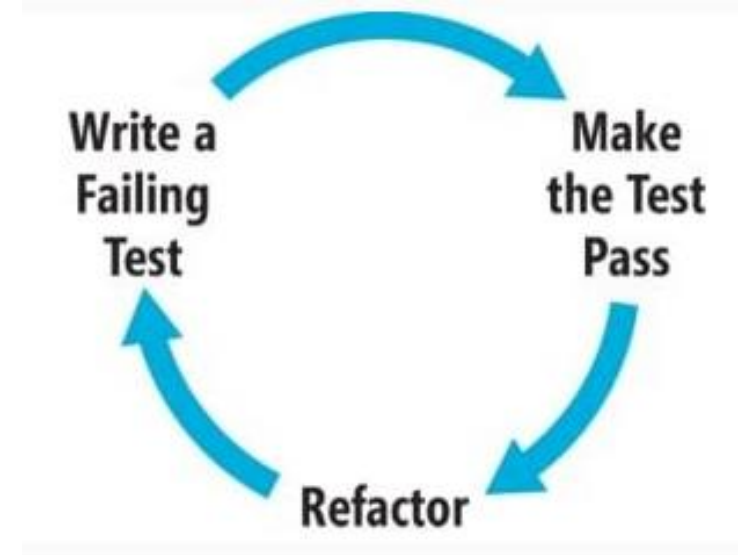
Techniques de test logiciel

- Techniques
 - Dynamiques/statiques
 - Génération de tests
 - Fonctionnels/structurels
 - Plusieurs niveaux de tests :
 - Unitaire, intégration, système, non-régression
- 

TDD - Développement Dirigé par les Tests

C'est le processus d'implémentation de code en écrivant d'abord des tests. Le cycle est le suivant :

1. Écrire un seul test qui décrit une partie du problème à résoudre ;
2. Vérifier que le test échoue, autrement dit qu'il est valide, c'est-à-dire que le code se rapportant à ce test n'existe pas ;
3. Écrire juste assez de code pour que le test réussisse ;
4. Vérifier que le test passe, ainsi que les autres tests existants ;
5. Remanier le code, c'est-à-dire l'améliorer sans en altérer le comportement.



TP Tests unitaires

<https://github.com/Cherfalyes/GLA-2022/>

References

- <http://david.roumanet.free.fr/BTS-SIO/SLAM5/UML%20diagramme%20de%20classe.pdf>
- <https://mrproof.blogspot.com/2012/10/exercice-corrige-uml-diagramme-de.html>
- <https://www.omg.org/spec/OCL/2.4/PDF>
- <https://realpython.com/python3-object-oriented-programming/>
- <http://niedercorn.free.fr/iris/iris1/uml/umltd5.pdf>
- <https://latavernedutesteur.fr/2017/11/03/les-niveaux-de-test/>
- https://www.fil.univ-lille1.fr/~wegrzyno/portail/Info/Doc/HTML/tp_conditionnelle.html
- <https://code.tutsplus.com/tutorials/beginning-test-driven-development-in-python--net-30137>