Qualité fonctionnelle

- Vérifier qu'un (sous-)programme réagit de la façon prévu par les spécifications : **valider un niveau de qualité en accord avec les attentes métier (juridique...) et technique**.
- 3 clés d'un test unitaire en boite noire : les *données en entrée*, le *(sous-)programme à tester* et les *résultats attendus*.
- Un (sous-)programme est testable *automatiquement* (≈en boite noire) seulement si on peut récupérer les informations nécessaires pour déterminer son bon déroulement.
- **Un test réussi seulement si les résultats du (sous-) programme sont conformes aux résultats attendus**.
- Un test ne permet pas d'identifier la cause d'une erreur, ni de la réparer : on mesure la qualité à un instant t.

Conception des tests unitaires

Ordonnancement des tests :

- Les cas généraux avant les cas particuliers.
- Les cas les plus prioritaires/critiques avant les autres.
- Si un (sous-)programme A dépend d'un sous-programme B alors il faut tester B avant A.

Caractéristiques des tests unitaires

2 objectifs des tests unitaires : **couverture du code** testé et **couverture des données** testées.

- Un test unitaire doit s'exécuter rapidement (< quelques secondes).
- Un test unitaire doit être indépendant des autres tests.
- Un test unitaire doit être répétable.
- Un test unitaire doit se valider/s'invalider lui-même.

Conception d'un test unitaire : mauvais exemple

```
/*
Nom du test : test_au_carre
Description brève : vérifie que la fonction au carre fonctionne pour
                    un double <0, nul et >0 ; vérifie que le résultat
                    est valide

Précondition(s) : la fonction au_carre est bien definie
Postcondition(s) : le test se termine uniquement si le test réussi.
*/
void test_au_carre(void){
   assert( au_carre( 0. ) == 0.0 ) ;

   assert( au_carre( 1. ) == 1.0 ) ;
   assert( au_carre( 2. ) == 4.0 ) ;
   assert( au_carre( 3. ) == 9.0 ) ;
   assert( au_carre( 3000000000. ) == 9e+18 ) ;

   assert( au_carre( -1. ) == 1.0 ) ;
   assert( au_carre( -2. ) == 4.0 ) ;
   assert( au_carre( -3. ) == 9.0 ) ;
   assert( au_carre( -3000000000. ) == 9e+18 ) ; }

int main(void){
   test_au_carre() ;
   test_au_cube() ;  /* ici 2 tests sont lancés ensembles... */
   ... }
```

Conception d'un test unitaire en boite noire

Un test doit pouvoir être lancé indépendamment des autres tests.

Une solution adaptée consiste à :

- **écrire un fichier source par test**, le nom du fichier source permettant d'identifier le test de façon unique.
- **mettre les intructions du test dans la fonction** main **du fichier source.**

Conception d'un test unitaire : bon exemple

```c
/*
Nom du fichier : test_au_carre.c
*/
#include <assert.h>
#include "au_carre.h" /* on inclut le module utilisateur contenant la fonction au_carre */

int main(void){
   assert( au_carre( 0. ) == 0.0 ) ;

   assert( au_carre( 1. ) == 1.0 ) ;
   assert( au_carre( 2. ) == 4.0 ) ;
   assert( au_carre( 3. ) == 9.0 ) ;
   assert( au_carre( 3000000000. ) == 9e+18 ) ;

   assert( au_carre( -1. ) == 1.0 ) ;
   assert( au_carre( -2. ) == 4.0 ) ;
   assert( au_carre( -3. ) == 9.0 ) ;
   assert( au_carre( -3000000000. ) == 9e+18 ) ;

   return 0 ;
}
```