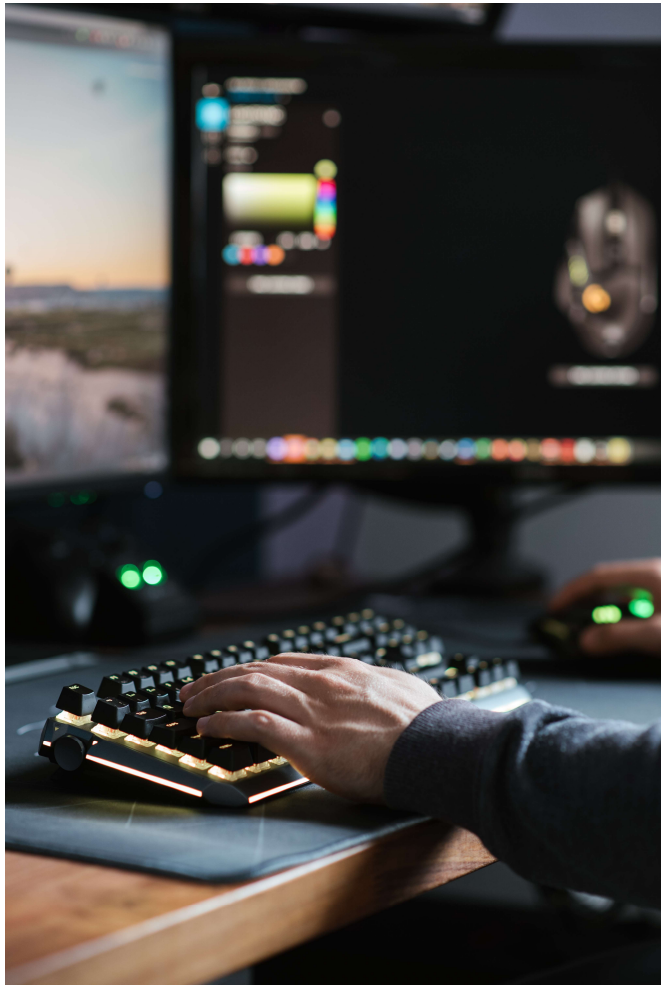
# Tester une application

# Qualité logicielle

**Qualité logicielle =**

- qualité de l'organisation et choix de méthodologie : cascade, cycle en V, agilité, hybride, suivi du projet, documentation
- qualité de l'environnement technique : langages, base de données, IDE (environnement de développement intégré), frameworks ...
- qualité de la conception et des développements : normes (RFC ...), revue de code, pair programming, éléments de sécurité ...
- qualité des tests : types de tests, pertinence, automatisation, plateformes ...



# 2 familles de tests

On distingue 2 familles de tests :

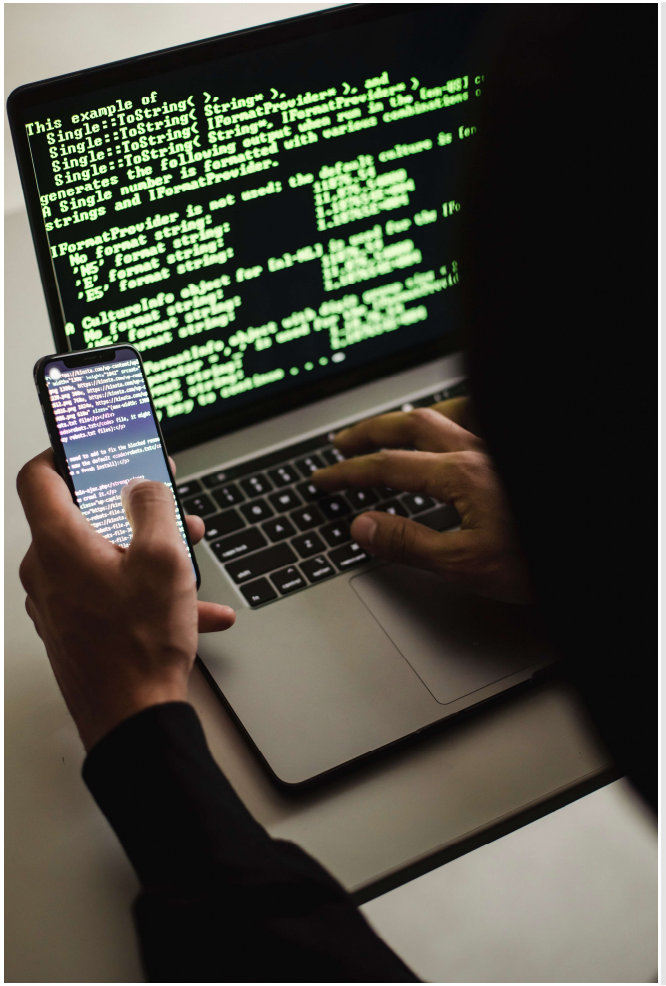
- les tests de type boîte blanche effectués en toute connaissance du code
- les tests de type boîte noire :
  - effectués sans avoir connaissance du développement
  - effectués par des testeurs externes
  - effectués par les utilisateurs de l'application

Les types de tests que nous allons détailler dans la suite ce support sont parfois communs à ces deux familles

# Tests unitaires

Ils valident des structures au niveau du code : méthodes, classes…

- il faut impérativement connaître le code à tester
- ils ne peuvent qu'être exécutés par le développeur ou un membre de l'équipe de développement
- exemples :
  - vérification de toutes les valeurs acceptées par une méthode
  - vérification d'un calcul



# Tests unitaires

# Exemple

```php
<?php
function testAdditionDeDeuxEntiers() {
   $result = additionner(5, 2);
   assert($result === 7, 'La fonction additionner ne retourne pas le résultat attendu.');
}

function additionner($a, $b) {
   return $a + $b;
}

// on appelle le test
testAdditionDeDeuxEntiers();
   echo 'Test passé';
?>
```

- la fonction **testAdditionDeDeuxEntiers** utilise la fonction **additionner** avec les paramètres 5 et 2
- on vérifie si le résultat est bien 7
- si la condition n'est pas vérifiée, on affiche un message d'erreur
- pour finir, on signale que le test a été exécuté

# Tests d'intégration → Tests de bout en bout

- plusieurs parties d'une application peuvent être développées séparément
- l'intégration est la phase de regroupement de ces développements séparés
- les tests d'intégration vérifient la cohésion (cohérence) du code fourni par des développeurs différents
- ils valident aussi la communication entre ces développements. Exemple : un développeur a codé la connexion à un site Web, un autre a codé la page d'accueil
- ces tests d'intégration vont nous amener vers les tests de bout en bout, " end to end "

# Tests d'intégration → Tests de bout en bout

# Illustration : le fonctionnement de mon bras

Nous pratiquons 3 **tests unitaires**

- le poignet fonctionne
- le coude aussi
- l'épaule également

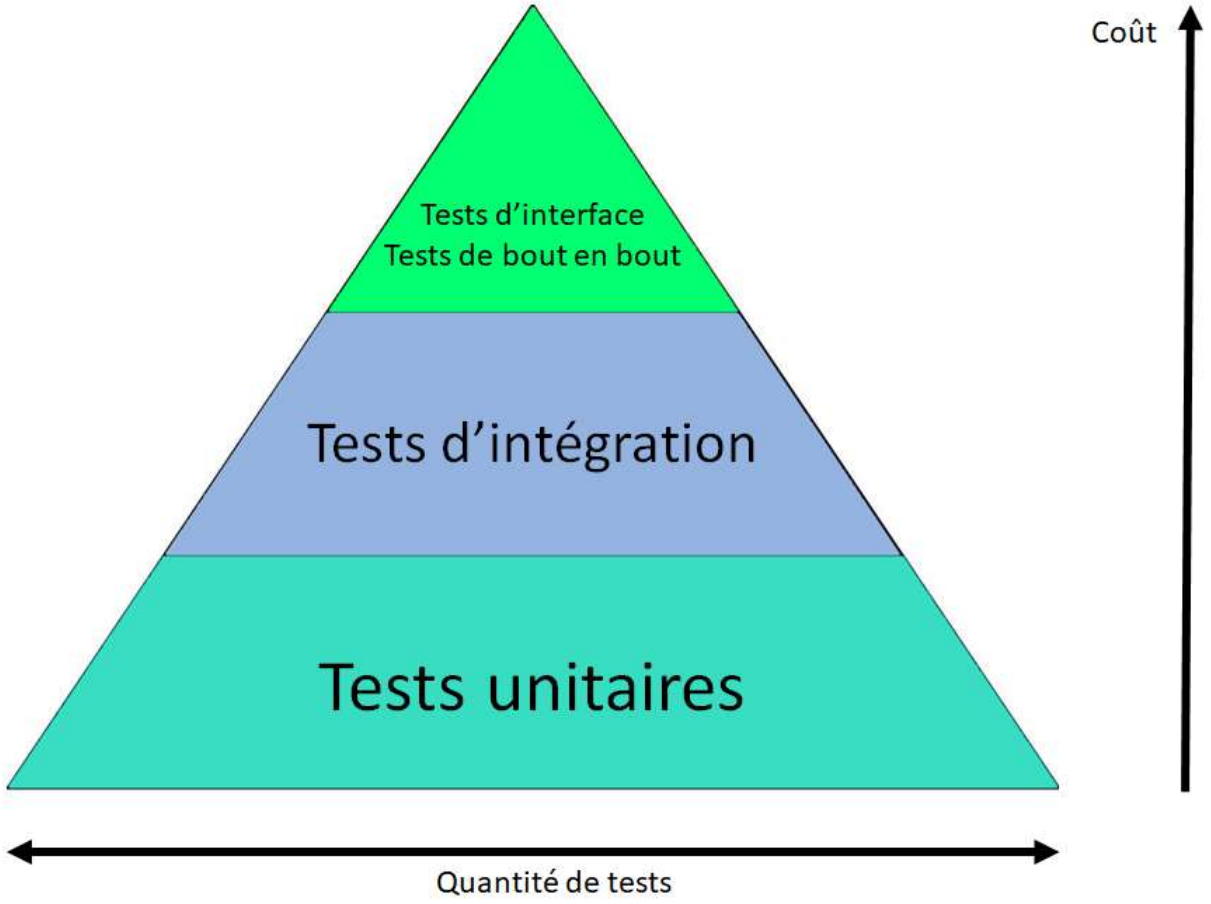
Puis nous avons le **test d'intégration**

en activant simultanément poignet, coude et épaule, je constate que l'ensemble de mon bras est opérationnel

Et enfin voici le **test de bout en bout**

comme mon bras fonctionne, je m'en sers par exemple pour ouvrir une porte … ou me protéger des rayons du soleil

# Pyramide des tests



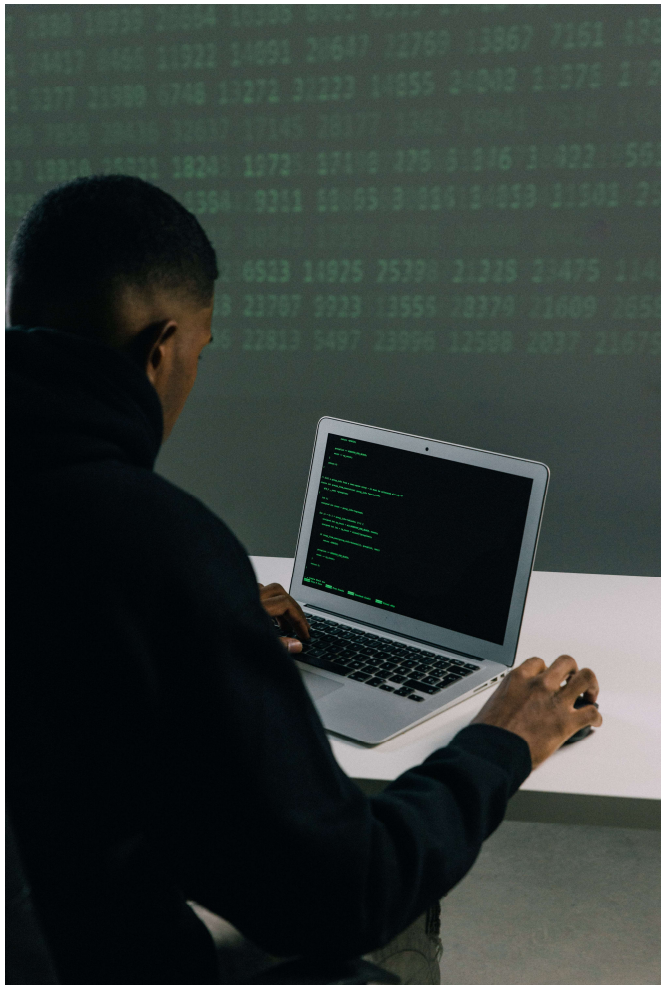
# Tests fonctionnels

Ils permettent de valider une fonctionnalité conformément aux spécifications et aux attentes de l'utilisateur

Exemples :

- la connexion d'un utilisateur à son profil dans une application web
- l'impression d'un document
- la validation d'un panier sur un site de type e-commerce
- le calcul d'une facture

Ils sont pratiqués par les développeurs et / ou les utilisateurs



# Tests de performances = Tests de robustesse

Ces tests évaluent la **réactivité**, la **vitesse** et la **stabilité** d'une application sous différentes conditions d'exécution et de charges

- exemple de réactivité : tests de **temps de réponses** à une requête SQL en base de données
- exemple de vitesse : tests de **temps de chargement** d'une page web
- exemple de stabilité : **tests en volumes** (montée en charge) :
  - simuler l'évolution des volumes d'une base de données et s'assurer du bon comportement de l'application
  - simuler un nombre élevé de connexions simultanées à un site web

**Objectif : on ne veut pas de dégradation du service**

# Tests de non-régression

- Les tests de no-régression valident que des fonctions déjà développées soient toujours fonctionnelles lors d'une nouvelle version de l'application
- Scénario de non-régression :
  - la "version 1" d'une application possède des tests qui ont permis de la valider et de la livrer au client
  - on développe une "version 2"
  - on doit impérativement rejouer les mêmes tests de la "version 1"
  - en livrant la "version 2", on aura donc la certitude que les fonctions de la "version 1" sont toujours opérationnelles

# Tests utilisateurs

## Tests d'acceptation utilisateur

- UAT = User Acceptance Test
- les utilisateurs s'assurent que le l'application répond à leurs besoins

## Tests de convivialité

Les utilisateurs évaluent la facilité d'utilisation des interfaces (ergonomie) et expriment leur niveau de satisfaction

# Tests divers

## Tests d'exploration

- les testeurs découvrent et apprennent l'application en la testant

## Tests contextuels

- ils nécessitent une bonne connaissance du contexte " métier "
- on parle aussi de connaissance culturelle
- exemples : un médecin teste un logiciel scientifique, un gamer teste un jeu ...

# Tests divers

**Tests comparatifs**

- on compare points forts et points faibles de plusieurs applications similaires
- on note la compétitivité des produits

**Tests " alpha "**

- effectués lorsque le développement est presque achevé
- on accepte encore de faire des changements mineurs

**Tests " bêta "**

- effectués lorsque le développement et tous les autres tests sont terminés
- l'objectif est de trouver les derniers bugs avant de déployer en production la version finale du logiciel

# Les 5 environnements

**1 – l'environnement local**

- l'ordinateur du développeur
- contient toutes les dernières versions de notre propre code
- chacun teste ses nouvelles fonctionnalités avant de les tester dans les environnements suivants

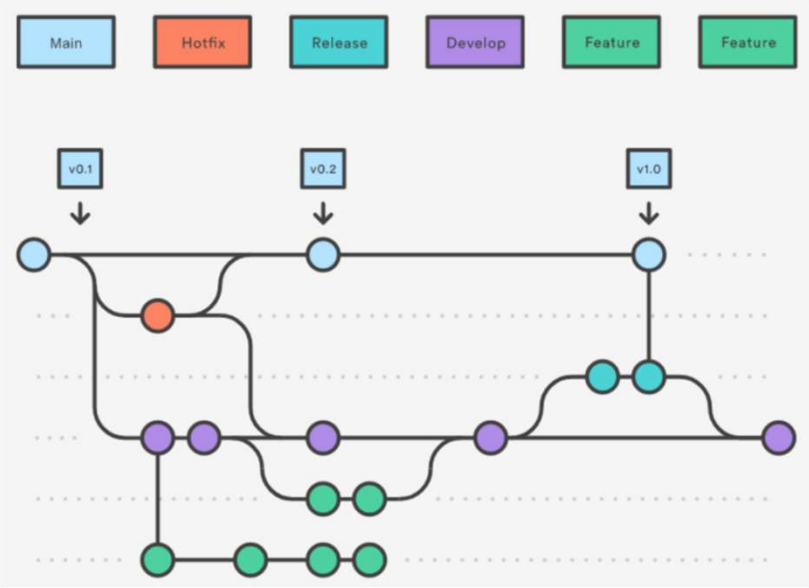
**2 – l'environnement de développement**

- les codes développés dans l'environnement local sont poussés sur un serveur commun
- on teste dans un environnement serveur (local ou en ligne)
- on corrige les bugs avant de diffuser le code sur les environnements suivants

# Les 5 environnements

## 3 – l'environnement de testing

- on teste les développements effectués en parallèle sur les différentes branches
- on a la souplesse de passer d'une branche à une autre pour tester les différentes versions du code

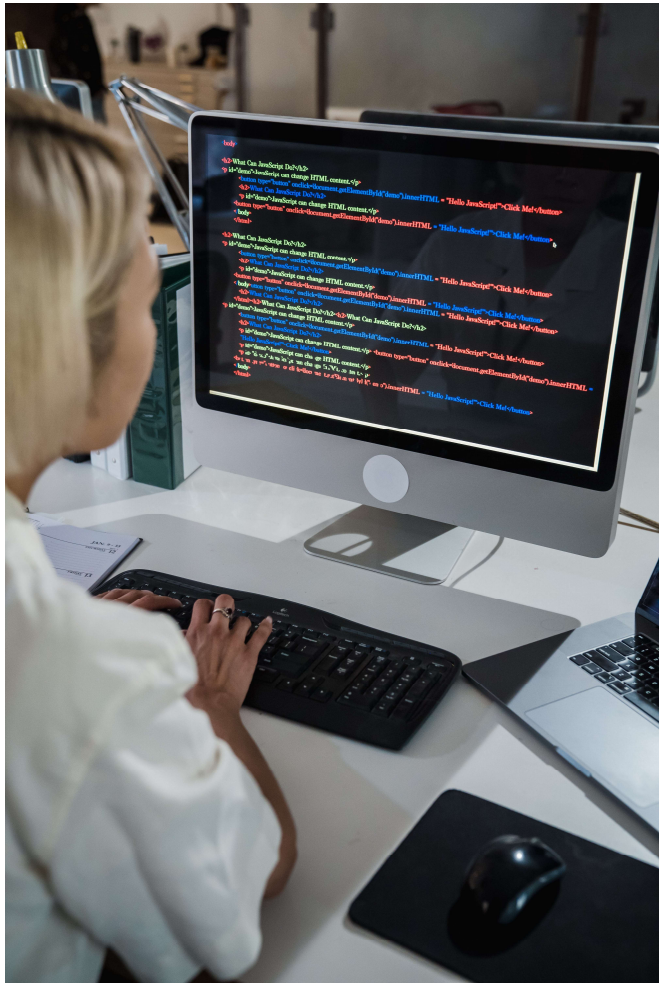


# Les 5 environnements

## 4 – l'environnement de staging (aussi appelé **pré-production)**

- c'est le dernier environnement avant la livraison
- il est identique à la production
- c'est un environnement de validation finale

On s'assure que tout fonctionne correctement avant de déployer une nouvelle version ou des mises à jour  pour les utilisateurs finaux sur l'environnement de production



# Les 5 environnements

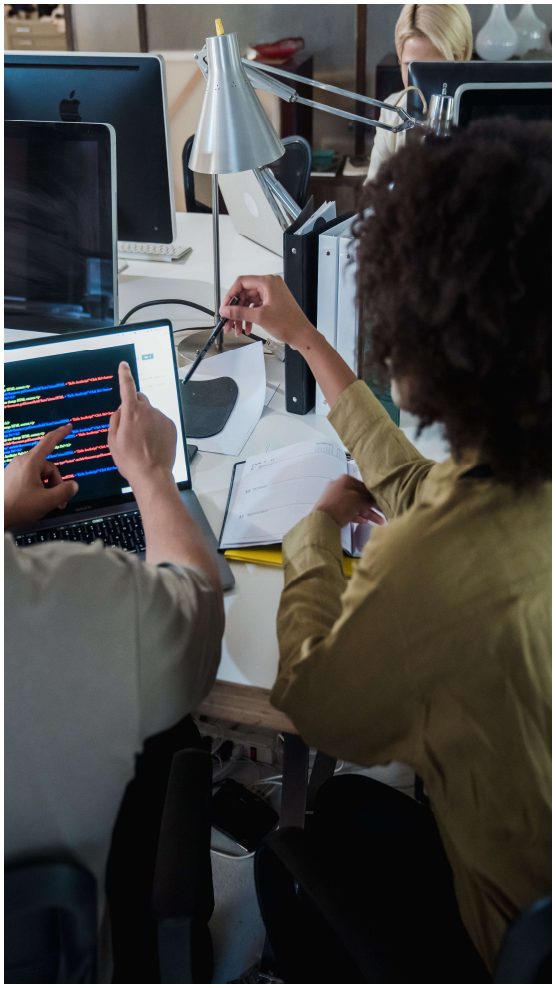
## 5 – l'environnement de production

- il accueille la version finalisée du logiciel
- c'est le seul environnement accessible aux utilisateurs finaux

---

au minimum : on aura 3 environnements

local + staging + production

au maximum : on pourra utiliser 5 environnements

local + développement + testing + staging + production