Test et TDD

Les tests Unitaires (Unit tests)

Unit tests only test a single part of your implementation.

- **Quel est son rôle ?**

un test unitaire permet de tester le bon fonctionnement d'une partie précise d'un programme. Il permet de s'assurer que le comportement d'une application est correct. Par exemple, imaginez une application mobile qui vous permette de gérer un porte-monnaie virtuel. Vous testeriez si l'euro que vous mettez dedans soit bien comptabilitsé. Vous testeriez également si vous pouvez récupérer l'argent qui s'y trouve.

- **Quel est son intérêt ?**
- On identifiera donc potentiellement un défaut à un endroit précis (car vérification sur section précise).
- être confiant que le code marche
- Make better architectural decisions (platform/modules/structure⟶ = donner à chaque bout de code une tâche précise et pas plusieurs)
- Il vous permet d'avoir un retour rapide sur votre développement. Si le test échoue, vous pouvez rapidement corriger votre code pendant que tout est frais dans votre esprit ;
- Il vous permet d'avoir un filet de sécurité. Si vous cassez quelque chose, vous vous en apercevez tout de suite ;
- Il vous permet d'éviter d'accumuler de la dette technique, en ayant une application qui soit facilement maintenable et évolutive.
- **A-t-elle des désavantages ?**
- **Y a-t-il plusieurs façons de s'en servir ?**

L'idéal consiste à écrire des tests de chaque catégorie. La recommandation est la suivante :

- 70% de tests unitaires ;
- 20% de tests d'intégration ;
- 10% de tests d'interface.
- **Quelles sont les étapes importantes pour la mettre en place ?**

No dependencies or integrations, no framework specifics. They're like a method that returns a link in a specific language:

```
export function getAboutUsLink(language){
  switch (language.toLowerCase()){
    case englishCode.toLowerCase():
      return '/about-us';
    case spanishCode.toLowerCase():
      return '/acerca-de';
  }
  return '';
}
```

Des frameworks de test: Jest, Mocha pour JS

- **Existe-t-il des contextes (langages, environnements, outils) où elle n'existe pas ?**
- **Quelles sont ses alternatives ?**

Les tests d'intégration (integration tests)

At some point, your code communicates with a database, file system or another third party. It could even be another module in your app.

That piece of implementation should be tested by integration tests.

//⟶il s’agit ici de tester le système au global, c’est-à-dire en intégrant tous ses composants.

Les tests d'intégration, qui permettent de tester plusieurs composants. Certains sont simulés, d'autres sont réels. Ils peuvent s'exécuter sur l'émulateur ou sur un équipement réel. Ils sont plus longs à écrire, mais sont beaucoup plus fidèles que les tests unitaires ;

- **Quel est son rôle ?**
- **Quel est son intérêt ?**
- **A-t-elle des désavantages ?**
- **Y a-t-il plusieurs façons de s'en servir ?**
- **Quelles sont les étapes importantes pour la mettre en place ?**
- **Existe-t-il des contextes (langages, environnements, outils) où elle n'existe pas ?**
- **Quelles sont ses alternatives ?**

Les tests de parcours/validation

Functional tests

Unit tests and integration tests give you confidence that your app works. Functional tests look at the app from the user's point of view and test that the system works as expected.

//Les tests d'interface, qui permettent de tester une application comme un vrai utilisateur. Ces tests ont l'avantage d'être extrèmement fidèles à la réalité, mais ce sont les plus longs et les plus compliqués à écrire.

- **Quel est son rôle ?Quel est son intérêt ?**
- **A-t-elle des désavantages ?**
- **Y a-t-il plusieurs façons de s'en servir ?**
- **Quelles sont les étapes importantes pour la mettre en place ?**

- **Existe-t-il des contextes (langages, environnements, outils) où elle n'existe pas ?**
- **Quelles sont ses alternatives ?**

Qu’est ce que le code coverage

- **Quel est son rôle ?Quel est son intérêt ?**

Code coverage is the percentage of code which is covered by automated tests. Code coverage measurement simply determines which statements in a body of code have been executed through a test run, and which statements have not.

- it is generally accepted that 80% coverage is a good goal to aim for. Trying to reach a higher coverage might turn out to be costly, while not necessary producing enough benefit.

- **A-t-elle des désavantages ?**

The first time you run your coverage tool you might find that you have a fairly low percentage of coverage. **If you're just getting started with testing it's a normal situation to be in and you shouldn't feel the pressure to reach 80% coverage right away. → Because tests should be written**

- **Y a-t-il plusieurs façons de s'en servir ?**

To calculate the code coverage percentage, simply use the following formula:

Code Coverage Percentage = (Number of lines of code executed by a testing algorithm/Total number of lines of code in a system component) * 100.

The common metrics that you might see mentioned in your coverage reports include:

- **Function coverage:** how many of the functions defined have been called.
- **Statement coverage:** how many of the statements in the program have been executed.
- **Branches coverage**: how many of the branches of the control structures (if statements for instance) have been executed.
- **Condition coverage:** how many of the boolean sub-expressions have been tested for a true and a false value.
- **Line coverage:** how many of lines of source code have been tested.

- **Quelles sont les étapes importantes pour la mettre en place ?**

Use a code coverage tool

- **Existe-t-il des contextes (langages, environnements, outils) où elle n'existe pas ?**
- **Quelles sont ses alternatives ?**

Qu’est ce que le TDD (test driven development)

- **Quel est son rôle ?Quel est son intérêt ?**

Il existe trois approches concernant les tests unitaires :

1. Les développeurs n'en écrivent jamais (ne riez pas, c'est malheureusement la majorité des cas) ;
2. Les développeurs écrivent les tests quand ils ont le temps et la motivation (c'est à dire très rarement) ;
3. Les développeurs commencent par écrire les tests avant toute ligne de code métier.

Cette dernière approche porte un nom : le **TDD**, pour Test-Driven-Development.

→La règle est assez simple : vous commencez par écrire les tests qui permettent de valider les différentes exigences de votre application. Ensuite, vous écrivez le code de votre application qui permet de valider ces tests.

→l'approche TDD présente certains avantages : moins de code à écrire, une application plus fiable.

- **A-t-elle des désavantages ?**

il n'est pas toujours évident d'avoir cette approche avec des tests d'interface ou lorsqu'une connectivité réseau est présente. Car les paramètres qui rentrent en compte sont multiples.

- **Y a-t-il plusieurs façons de s'en servir ?**

exemple pratique

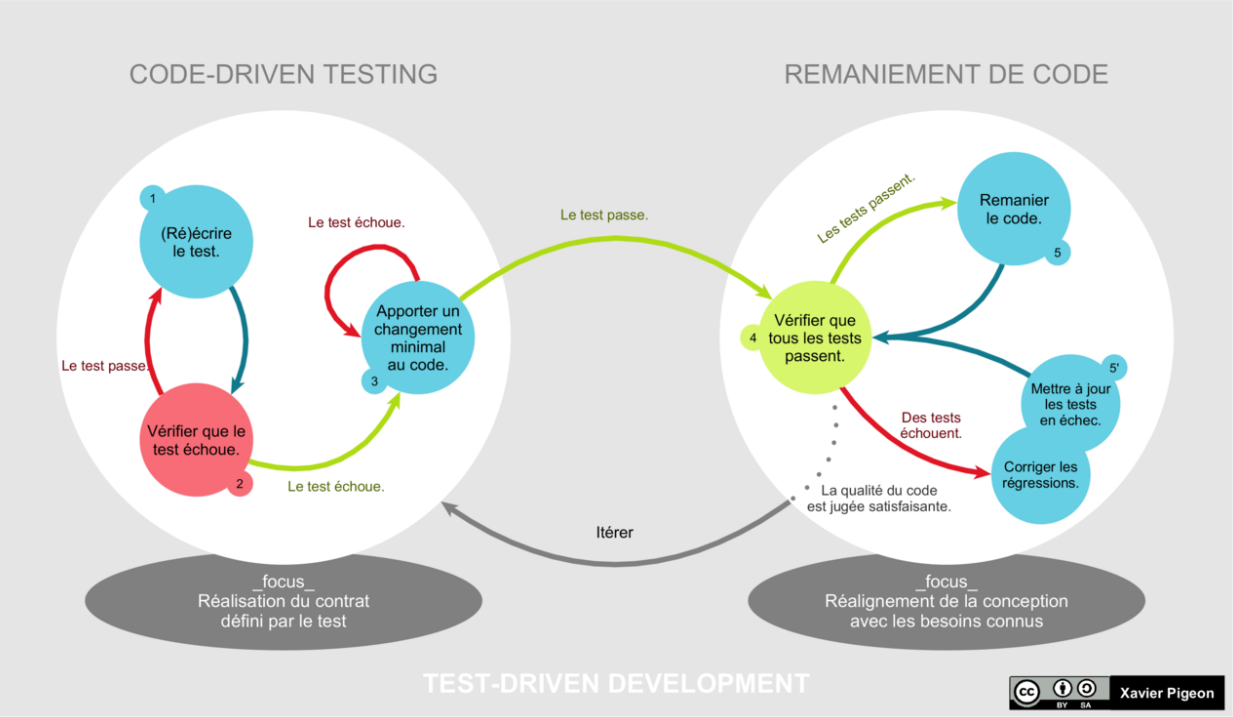
https://grafikart.fr/tutoriels/tdd-pratique-javascript-1205

- **Quelles sont les étapes importantes pour la mettre en place ?**

Le processus préconisé par TDD comporte cinq étapes :

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. puis remanier le code, c'est-à-dire l'améliorer sans en altérer le comportement.

Ce processus est répété en plusieurs cycles, jusqu'à résoudre le problème d'origine dans son intégralité. Ces cycles itératifs de développement sont appelés des micro-cycles de TDD.



- **Existe-t-il des contextes (langages, environnements, outils) où elle n'existe pas ?**
- **Quelles sont ses alternatives ?**

A quoi doit répondre un test unitaire (F.I.R.S.T.)

F.I.R.S.T. stands for Fast, Independent, Repeatable, Self-Validating and Timely.

- **Quel est son rôle ?**

These principles help to write well-crafted unit tests and will make testing easier.

- **Quel est son intérêt ?**

The purpose of these principles is to create clean code)(easy to understand and reuse), and by extension clean tests (easy to change and reuse). Clean tests help unit tests to keep the production code flexible, maintainable and reusable.

- **A-t-elle des désavantages ?**

- **Y a-t-il plusieurs façons de s'en servir ?**
- **Quelles sont les étapes importantes pour la mettre en place ?**

FAST

- unit tests should be fast (< 100ms / test) and help to localize problems. In order to be fast, they should not talk to a database, communicate across a network, touch a file system, or run in a special configuration setup.
- Here is a story showing why tests should be executed quickly:

1. If tests are slow, we will not run them frequently ;
2. If they are not run often, we will miss bugs ;
3. If they take too long to find, we will struggle to maintain our test code .... etc.

- How to write fast tests :
  - One assert per test (en français : une déclaration ??)
  - variables and functions should be short and descriptive

INDEPENDENT

Tests isolate failures means that tests should not depend on each other in order to prevent a single failing test to cause a succession of other failing tests. → If a test cannot be independent, it means it is not short and specific enough.

How should we write Independent tests? Build-operate-check pattern :

BUILD-OPERATE-CHECK Pattern

The BUILD-OPERATE-CHECK pattern helps writing more independent tests. In a test function:

- the BUILD part is about building the input data so they are not shared between tests ;
- The OPERATE part is about calling the method we want to test. It clarifies what we are testing ;
- The CHECK part is about asserting the result of the operation.

REPEATABLE

- Tests should be repeatable in any conditions / environments such as production environnement, QA environnement, no network, or different day, place, time...

SELF-VALIDATING

Tests should be self-validating, meaning that we should not go through a log file or compare data to know if they pass.

TIMELY

We should write them from the beginning (TDD method)

- **Existe-t-il des contextes (langages, environnements, outils) où elle n'existe pas ?**
- **Quelles sont ses alternatives ?**