Le podcast « Artisan développeur » a été un tremplin vers ma deuxième vie professionnelle dans le sens où il m’a redonné goût à l’informatique, et surtout redonné l’envie d’apprendre, de découvrir des choses que je n’avais pas eu le temps de découvrir par manque de temps.

Un des sujets abordé dans tous les sens dans le podcast est celui des tests unitaires et du TDD. Je pense faire partie d’une génération qui n’a pas connu ces concepts, ni au cours de ma formation initiale, ni dans les entreprises que j’ai fréquentées.

Le fait d’écouter les podcast et les intervenants parler de ce sujet régulièrement m’a donné le goût de creuser le sujet. Mais par où commencer ? Le concept me semblait relativement simple mais quand tu n’as jamais écrit le moindre test unitaire de ta vie, même quelque chose de simple est un saut dans l’inconnu.

Le TDD tel que je l’ai compris est une pratique de développement orienté par les tests (comme son nom l’indique TDD = Test Driven Developement). Le TDD n’est pas une méthode de test mais une méthode de développement.

* On commence à écrire un test, le système renvoi un drapeau rouge (le test ne passe pas car on n’a pas encore écrit de code).
* On écrit le code le plus simple pour faire passer le test, le système renvoi un drapeau vert
* On améliore la qualité du code si l’on pense que c’est nécessaire (refactoring).

Et on répète l’opération jusqu’à apporter une solution au problème initial. Les tests sont de taille minimale et doivent rapidement passer au vert (on parle de petites étapes ou baby step).

Je pense avoir compris le concept, j’ai écouté pas mal de podcast, regardé quelques vidéos pour voir comment font les experts du domaine.

Je choisis une liste de Katas (des petits exercices) que je trouve sur le site Web de Ippon.

[Mon catalogue de katas (ippon.fr)](https://blog.ippon.fr/2021/04/12/mon-catalogue-de-katas/)

Je choisis un langage simple que je connais un peu, le PHP et sa bibliothèque de test PHPUnit (que je n’ai jamais utilisé). Mon quotidien tourne autour du PLSQL mais je préfère choisir un langage un peu plus rependu qui dispose d’une bibliothèque de tests. Et puis les tests unitaires en PLSQL, pour le coup c’est un grand saut dans l’inconnu et je risque de me heurter à un manque de ressources et une faible communauté pour m’aider.

Je me rappelle avoir commencé par le kata Fizzbuzz, car j’en avais entendu parler par Mickael Azerhad, un spécialiste du sujet. J’avais l’algorithme en tête assez vite sans TDD mais je voulais voir comment le mettre en place en utilisant TDD. La veille d’écrire mes premiers tests sur l’exercice, je me souviens avoir regardé la vidéo de Mickael, dans laquelle il montre étape par étape, comment il procède, avec sa méthodologie, sa façon de découper le problème par petites étapes, et de ne volontairement pas aller trop vite à la solution. La façon dont il résout le problème est hyper intéressante, en prenant le temps d’expliquer étape par étape sa démarche.

Le lendemain, je réussi l’exercice sans trop de difficulté mais comme j’avais vu une vidéo et que j’avais une idée de l’algorithme en tête, je n’ai fait qu’effleurer la puissance de cette pratique de développement qu’est le TDD.

J’ai continué avec les katas proposés, par la suite plus dans un mode découverte. Je me rappelle avoir réalisé des exercices sur les années bissextiles et les chiffres romains.

Mais cette fois ci, sans avoir réfléchi à l’avance sur le comportement de l’algorithme avant de commencer. Les années bissextiles est un exercice très simple mais l’idée de réaliser une solution qui répond à l’énoncé sans avoir à réfléchir à l’architecture du code, est quelque chose que je n’avais jamais fait auparavant. Et en me laissant guider par les tests j’arrivais à la solution rapidement. Le fait de ne pas se poser pour réfléchir à comment faire, permet à mon sens de libérer un peu sa charge mentale.

Cela devenait encore plus intéressant pour des exercices plus complexes. Pour lesquels je n’avais pas la moindre idée de la solution à adopter, ni du chemin à prendre pour les résoudre, mais juste en écrivant les bons test, j’arrivais au résultat de manière surprenante en écrivant un code très propre.

En parlant de code propre, une des forces du TDD, c’est du pouvoir refactorer le code sans risque, à savoir qu’à la moindre erreur, les tests te disent que le code ne répond plus aux tests. En temps normal, en testant manuellement le code, je serai probablement passé à côté de cas à la marge. Mais en relançant les tests, on voit tout de suite si le code n’est pas cassé (c’est rouge ou vert, facile !).

Le fait de refactorer aussi facilement m’a permis d’écrire un joli code, alors qu’avant de faire le kata, je n’avais aucune idée des solutions aux problème que je devais résoudre, ni de la structure du code que j’allais écrire.

Jusqu’à présent j’ai surtout mis en place les tests automatiques pour des katas ou des projets personnels. Mais le fait d’avoir essayé m’a permis de comprendre l’intérêt et les avantages d’utiliser ce genre d’outil.