Notes sur le module : *Refactoring*.

Armel Pitelet

13 février 2022

Table des matières

On vas utiliser https://github.com/nicolas63/GildedRose (repos de Nico) et https://github.com/emilybache/GildedRose-Refactoring-Kata (autre repo avec d'autre languages) pour s'entrainer à refactorer du code. L'idée est de partir d'un code Legacy et de l'étendre. C'est un Kata de refoctaring (à Googler, avec Coding Dojo.)

Note :
Sur le repos de Nico on peut passer le code en .Net Core.

Remarque : *Technique du GoldenMaster.*
On commence par déporter le code legacy, pour pouvoir comparer la sortie du code modifié et de l'ancien code. Cela nécessite un jeu de test suffisant qui couvre l'intégralité des sorties du code (il faut être confiant dans les données de prod). Dans cette technique on va suffixer les classes Legacy avec *GoldenMaster*.

Remarque : *Une autre manière de faire.*
Commencer par du rétro ingéniéring pour récupérer les specs de chaque fonctionnalités. On code tt les tests associés, puis on commence les modifications. La différence avec avant est que l'on test les fonctionnalités une par une et l'on ne test plus la sortie globale du code.

Note :
Note Should().BeEquivalentTo() compare bien les valeurs des propriétés en plus de la structure des objets. Voir quand même le **site**.

Remarque : *Lors des tests.*
Il faut s'assurer que le jeu de test soit exhaustif et que l'on itère bien de la même manière qu'en production.

Remarque : *Sur le code coverage et les Test.*
La version Pro de VS propose le code coverage dans les tests. Sinon il faut chercher une extension pour. On peut également trouver des extensions qui permettent de lancer les tests à chaque changement.

Remarque : *Sur le procédé de refacto.*
Il faut identifier les redondances, sortir les constantes et magics. Supprimer les négations dans les booléen, merger, flipper les if au besoin pour les trier et les séparer pour identifier les redondances et surtout la logique de chaque action. Il faut changer peut à peu le code pour aller vers l'objectif que l'on souhaite et commit à chaque fois. On veut des méthodes les plus petites possibles, des classes

les plus simples possibles.

Note :

Ces méthodes de refactoring permettent de s'arréter en cours de route, pour reprendre ensuite puisque tt est testé en permamence à chaque itération.