Architecture testable

On dit souvent que la pratique du TDD permet de laisser émerger l'architecture du logiciel, produisant du même coup une architecture testable et un code de meilleure qualité. À ce titre le TDD basé sur les tests unitaires servirait plus le design que le test en temps que tel. Certains vous diront que c'est l'inverse : si on écrit du code de bonne qualité alors il sera testable. D'autres encore vous diront que l'architecture produite par la pratique du TDD ne leur convient pas du tout. On liste ici des points sur lesquels la testabilité et la conception objet se rejoignent dans un objectif de code de qualité, quel que soit le bord par lequel on l'aborde.

Ce qui rend le code testable

SOLID est un acronyme proposé notamment par Bob Martin au début des années 2000, comme ligne directrice pour écrire du code fiable et robuste.

S Single Responsability Principle ou principe de responsabilité unique. Un objet fait exactement une seule chose, et est le seul à le faire. Ou encore : une classe doit avoir une seule raison de changer. Si ce n'est pas le cas, les tests couvrent trop de choses et deviennent lourds à maintenir, et on a tendance à les abandonner. De plus, plus les classes ont de responsabilité et plus la modification d'une des fonctionnalités va impacter les classes qui dépendent de cette classe mais n'utilisent pas la fonctionnalité.

O Open-Close Principle une classe doit être ouverte aux extensions et fermée aux modifications.

L Liskov Substitution Principle Une classe fille qui hérite d'une classe mère doit pouvoir être utilisée à la place de sa mère (par une autre classe) en toute transparence. Entre d'autres termes : la fille doit passer les tests que passent la mère.

I Interface Segregation Principle Plusieurs petites interfaces plutôt qu'une grosse : rejoint le SRP.

D Dependency Inversion Principle Dépendre d'abstraction et pas d'implémentations est essentiel pour permettre le test unitaire. On rend les dépendances explicites via l'**injection des dépendances**, ce qui permet de remplacer un objet du domaine par un mock ou autre objet du test.

En SVL on se focalise sur SRP et DI sous toutes ses formes, et on pratique sur différents cas d'études pour que ça vous rentre dans la tête.

Ce qui rend le code non testable

— Les `new` en dur dans le code (injecter une fabrique) ;
— toute forme d'état global : variables et méthodes statiques ;
— l'utilisation du pattern Singleton.