INF5153 – Génie logiciel : conception

Design for Testability

Jacques Berger

Objectifs

Introduire certains concepts pour favoriser la mise en test des classes

Prérequis

JUnit

Tests

Il existe plusieurs formes de tests, mais une seule est touchée par la conception : les tests unitaires

Test unitaire

Des classes avec des tests afin de vérifier le fonctionnement des autres classes

Voir exemples

Test unitaire

Un bon test unitaire est :

- Simple
- Court
- Indépendant des autres tests
- Indépendant de tout élément extérieur
- Rapide

Testabilité

La facilité avec laquelle nous pouvons effectuer des tests sur les classes du projet

Un couplage faible et une cohésion forte devrait améliorer la testabilité

Éléments de base

Éléments de base de la conception qui augmente la testabilité :

Fonctions courtes qui ne font qu'une chose

Classes avec une seule responsabilité

Éléments de base

Les méthodes qui retournent une valeur sont plus faciles à tester que les méthodes qui modifient l'état d'un autre objet

Si possible, favoriser les méthodes qui retournent une valeur

Dépendance

Le couplage complexifie la mise en test

Il peut être nécessaire de briser une dépendance avant la mise en test

Exemples de dépendance :

- L'opérateur new
- Appel d'une méthode statique

Injection de dépendance

L'injection de dépendance est une technique pour briser une dépendance dans une classe

Elle diminue le couplage en injectant la dépendance dans l'objet au lieu de laisser l'objet gérer la dépendance lui-même

Injection de dépendance

L'injection peut être faite avec un constructeur ou un setter

Ainsi, la création de la dépendance est déplacée au niveau de l'utilisateur de la classe plutôt que dans la classe elle-même

Voir un exemple

Mock object

Un mock object est un objet qui simule le comportement d'un autre objet

Il accumule de l'information sur la façon dont il est utilisé afin de tester les méthodes qui ne retournent pas de valeur

Mock object

Généralement utilisé pour simuler :

- Une base de données
- Un accès au disque
- Un accès réseau
- Un objet complexe

Mock object

Un mock object peut être réalisé par héritage ou par implémentation d'une interface commune

Voir des exemples

Mock object

La réalisation du mock object par interface est recommandée

Ça permet d'éviter les comportements par défaut de la classe de base

Mock object

Il est difficile de créer un mock object sur une classe qui interragit avec l'extérieur dans son constructeur

Un Wrapper pourrait être nécessaire dans ce cas

Voir un exemple

Mock object

Afin d'éviter ces gymnastiques complexes, il faut toujours concevoir nos classes afin de pouvoir facilement créer un mock object sur elle

Mock object

Ceci implique de :

Permettre un constructeur vide qui laisse l'objet dans un état nul

Ne pas interagir avec l'extérieur dans le constructeur vide

Mock object

Une classe qui fait de l'injection de dépendance pourra plus facilement recevoir un mock object pour la tester

Encapsulation

Les classes de tests sont habituellement dans le même package que la classe à tester

Les tests ont donc accès aux membres :

- public
- protected
- package-private

Mais les tests n'ont pas accès aux membres :

- private

Encapsulation

Faire des tests sur les membres privés en passant par les membres publiques peut être une tâche complexe

Il pourrait être utile de pouvoir accéder directement aux membres privés pour les tester

Encapsulation

Il peut être acceptable de briser l'encapsulation d'une classe et de changer la portée des méthodes private à protected ou package-private

Ainsi, les tests unitaires pourront y accéder

Perte d'encapsulation, meilleure testabilité

Static

Éviter les méthodes statiques

La dépendance vers une méthode statique est difficile à briser

La méthode statique ne peut pas être déplacée grâce à l'injection de dépendance, ni remplacée par un mock object

Static

Idéalement, puisqu'elles ne peuvent pas être remplacées, les méthodes statiques ne devraient pas communiquer avec l'extérieur

On s'en sert comme fonction utilitaire seulement

Instanciation

Certains objets peuvent être complexes à instancier

(ex. Un objet complexe prend un autre objet complexe en paramètre au constructeur)

Instanciation

Concevoir les objets pour qu'ils soient facile à instancier

Dans certains cas, il pourrait être acceptable de passer null en paramètre au lieu d'un objet (dépendamment du test à effectuer)

Conclusion

La testabilité est grandement influencée par la conception

Dans certains cas, il faut choisir entre une conception orienté-objet bien encapsulée ou une conception qui favorise la mise en test