Tests unitaires

Un test unitaire permet de s'assurer du bon fonctionnement d'une partie d'un programme (généralement d'une fonction, d'une procédure, d'un module ou d'un composant).

L'écriture d'un test permet de comparer une réalisation (implémentation) aux spécifications. Le concept de test unitaire n'est pas nouveau, et l'écriture de tests a longtemps été considérée comme une tâche secondaire.

La tendance s'inverse aujourd'hui, et la réalisation de tests devient centrale en conception logicielle, avec le développement combiné des framework xUnit facilitant leur mise en oeuvre, et l'apparition de l'extreme programming (XP).

Bonnes pratiques

Bonnes pratiques à suivre pour l'écriture de tests unitaires ou de composants :

1- Nommage

Nommer explicitement les cas de test et les tests de façon à rendre lisible leur rôle, en respectant des conventions de nommage.

- Préfixe (test)
- Nom de la méthode testée
- Scénario de test utilisé
- Comportement attendu quand le scénario est appelé

2- Couverture

Envisager tous les scénarios possibles, y compris les cas à la marge et pas uniquement le scénario principal correspondant au cas de succès.

3- Structure

Structurer les tests en respectant le modèle AAA ou protocole Given/When/Then :

- Arrange : Organisez vos objets, créez-les et configurez-les si nécessaire.
- Act : Agir sur un objet.
- Assert : Assertion de ce qui est prévu

4- Un cas/test

N'écrire qu'un seul cas par test, à moins qu'un déroulement soit imposé et qu'il faille vérifier les étapes intermédiaires.

5- Sans logique

Evitez d'introduire de la logique dans les tests (condition, itération...), pour ne pas introduire de bug, et pour

une meilleur lisibilité.

6- Sans dépendances

Eviter les dépendances avec d'autres modules/composant et utiliser les mocks (voir ci-dessous).

Tests avec dépendances Certaines unités à tester utilisent parfois d'autres services, non disponibles pendant la phase de test, trop complexes à créer, ou qui introduiraient un biais dans les résultats (exemple : connexion à une base de données, envoi de mail...).

Il est possible dans ce cas d'utiliser des doublures (mock objects) qui remplaceront les objets réels, en évitant les effets de bord.

*** Dummy**

Les dummy objects sont utilisés en lieu et place de paramètres requis mais ne sont jamais utilisés eux-mêmes.

*** Stubs**

Les stubs objects ont une implémentation qui retourne en dur des valeurs attendues pour le test.

*** Fake**

Les fake objects ont une implémentation respectant l'interface des objets dont ils sont la doublure, mais réduite et spécifique aux tests (une BDD en mémoire ou embarquée au lieu d'une connexnion à SGDB par exemple).

*** Spy**

Les spy objects sont des stubs fournissant des informations supplémentaires sur les appels effectués.

*** Mocks**

Les Mocks objects sont comme les Fake, avec les fonctionnalités du Spy, et intégrent en plus leurs propres assertions, sur les paramètres passés, les appels de méthodes...

Références

Mocks, fakes, stubs, dummy...

Librairies xUnit

- java : JUnit
- php : PHPUnit

Références

- Scott W. Ambler : une introduction au Développement Guidé par les Tests (TDD)
- Institut Agile : Developpement par les tests et présentation des principaux concepts liés