Tests unitaires

Nuwan Herath

2022-2023

Contexte

Tester ?

- Bonne pratique pour trouver et couper le mal à la racine avant qu'il ne cause de graves problèmes
- Economie de temps et d'argent

Objectifs des test

- Vérifier la conformité du logiciel
- Assurer la non-régression

Classification des tests selon la nature de l'objet

Classification liée au cycle de développement en V

- Test unitaire — test de composants
- Test d'intégration — test (d'intégration) technique
- Test système — test (d'intégration) fonctionnel ou homologation
- Test d'acceptation — test usine ou recette

Test unitaire

- Tester une unité de code
- Vérifier qu'une fonction réalise l'action souhaitée
- Ne pas dépendre de fonctionnalités extérieures

Test d'intégration

- Vérifier l'architecture du programme
- Vérifier que les fonctionnalités communiquent correctement

Test système

- Vérifier le bon déroulement d'un parcours utilisateur (user story)
- Réalisé par l'ordinateur ou un humain

Test d'acceptation

- Validation des fonctionnalités **avec** le client
- Retours par le client

Etre le plus exhaustif possible ?

> "Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence."
>
> — Edsger W. Dijkstra, *The Humble Programmer* (1972)

Que tester ?

Tester l'interface pas l'implémentation
Tester les méthodes publiques

De nombreux frameworks

Liste non-exhaustive

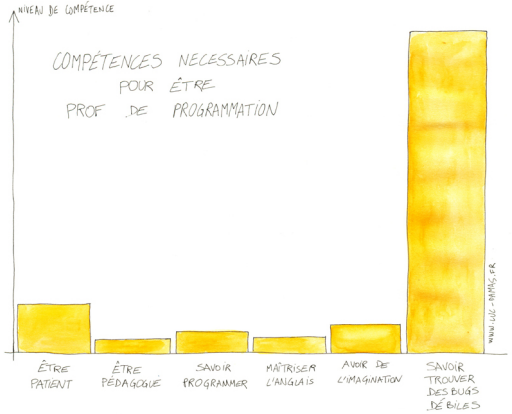
C	CUnit
C++	CppUnit
	CPUnit
	Google Test
Internet	Selenium
	Watir
Java	JUnit
JavaScript	JSUnit
	Unit.js
Perl	TAP
PHP	PHPUnit
Python	unittest
	Pytest

JUnit

JUnit est le premier framework a devenir populaire et a été développé par

- Erich Gamma $\longrightarrow$ design patterns
- Kent Beck $\longrightarrow$ eXtreme Programming

Mise en pratique


NIVEAU DE COMPÉTENCE
COMPÉTENCES NÉCESSAIRES POUR ÊTRE PROF DE PROGRAMMATION
ÊTRE PATIENT
ÊTRE PÉDAGOGUE
SAVOIR PROGRAMMER
MAÎTRISER L'ANGLAIS
AVOIR DE L'IMAGINATION
SAVOIR TROUVER DES BUGS DÉBILES
WWW.LOUC-EMMES.FR

Récupérer les fichiers sur Arche.

Mise en pratique

Python

L'assertion avec assert

```
def appliquer_reduction(prix, reduction):
    prix_reduit = prix - ( prix * reduction )
    assert 0 <= prix_reduit <= prix
    return prix_reduit
```

Voir 1-boutique.

L'assertion avec assert

Ajouter un message

```
def appliquer_reduction(prix, reduction):
    prix_reduit = prix - ( prix * reduction )
    assert 0 <= prix_reduit <= prix, "Le prix après réduction est négatif
    ↪  ou supérieur au prix initial."
    return prix_reduit
```

L'assertion avec `assert`

Usage

- Informe le développeur
- Ne signale pas une condition d'erreur attendue
- N'est pas une mécanisme de gestion d'erreurs d'exécution (`try ... catch`)

Introduction au module `unittest`

Découverte de quelques méthodes `assert`

Voir `2-calcul`.
Lire le contenu des deux fichiers, puis exécuter le test.

Convention

Les méthodes doivent commencer par `test`.

Remplacer

```
def test_somme(self):
```

par

```
def pas_test_somme(self):
```

puis relancer le test pour voir la différence.

Ignorer un test et lister les tests

Ignorer un test

Décorer l'une des méthodes avec

```
@unittest.skip("mon message")
```

Lister les tests

Obtenir plus d'information en passant en mode verbeux :

```
python3 test_calcul.py -v
```

Organiser la classe de test

On peut factoriser le code avec

- `setUp()` appelé **avant** l'exécution de chaque test,
- `tearDown()` appelé **après** l'exécution de chaque test.

Factoriser le code de test.

≪ Un aménagement de test (*fixture* en anglais) désigne la préparation nécessaire au déroulement d'un ou plusieurs tests, et toutes les actions de nettoyage associées. Cela peut concerner, par exemple, la création de bases de données temporaires ou mandataires, de répertoires, ou le démarrage d'un processus serveur. ≫

Plus de méthodes

Voir `3-personne`.
Lire le contenu des deux fichiers, puis exécuter le test.

Parenthèse

Une chaîne YouTube

https://www.youtube.com/@CodeAesthetic

Mise en pratique

Mise en pratique

Python (suite)

Pile - FIFO

TDD

Cette partie concerne la définition d'une structure Last In, First Out (LIFO) : une pile. Comme pour une pile d'assiettes par exemple, la dernière assiette rentrée est la première à ressortir.

Voir `4-pile`.

1. Executer les tests :

```
python3 -m unittest
```

2. Corriger les erreurs.
3. En utilisant la structure `list()`, implémenter les méthodes de la classe `Stack`.
4. Valider l'implémentation en vérifiant que tous les tests passent.

Pile - FIFO

Ecriture de tests

Comme dans tout projet, les besoins exprimés par le client changent en cours du projet, et il faut ajouter des méthodes :

- `size()` qui renvoie la taille de la pile ;
- `peek()` qui renvoie le prochain élément à dépiler, sans le retirer de la pile.

Développer cette extension de la classe en suivant un processus de développement piloté par les tests.

1. Ecrire d'abord les tests correspondants.
2. Ecrire ensuite les méthodes vides.
3. S'assurer que les tests échouent.
4. Modifier le code pour faire passer les tests.

Pile - FIFO

Ré-implémentation

Refaire la classe `Stack` sans utiliser de structure existante...
Pour cela, utiliser une liste chaînée définie manuellement.

- La classe `Element` qui contient un élément et une référence vers l'élément suivant (notion de chaîne).
- La nouvelle version de la classe `Stack` stockera une référence vers le premier objet, ce qui permet d'accéder au premier élément, de le dépiler, ou d'ajouter un nouveau élément en tête de la liste.

S'assurer de la non-régression du code : les tests doivent toujours passer avec la nouvelle implémentation.

Couverture des tests

La couverture est le pourcentage du code qui est testé.
C'est une métrique pour évaluer la qualité d'une suite de tests.
Il y a diverses manières de la mesurer (nombre de lignes de code, nombre d'instructions, nombre de branches...).

Couverture des tests

Présentation du module coverage

Installation

```
python3 -m pip install coverage
```

Usage

```
coverage run -m unittest
```

```
coverage report
```

```
coverage html
```

Couverture des tests

Exemple d'option du module coverage

```
coverage run --omit=nom_fichier -m unittest
```

```
coverage run --omit=nom_fichier report
```

```
coverage run --omit=nom_fichier html
```

Couverture des tests

Exploration du module coverage

Que signifie `-m` dans les deux instructions suivantes ?

```
coverage run -m unittest
```

```
coverage report -m
```

Consulter l'aide :

```
coverage run --help
```

```
coverage report --help
```

Mise en pratique

Mise en pratique

Google Test

Prérequis

Debian, Ubuntu

```
sudo apt install cmake
sudo apt install git
sudo apt install g++
```

Windows (PowerShell administrateur)

```
choco install cmake
choco install git
choco install g++
```

MacOS

```
sudo brew install cmake
sudo brew install git
sudo brew install g++
```

Exécution des tests

Construire le projet.

Unix

```
cmake -S . -B build
cd build
cmake --build .
```

Windows

```
cmake -S . -B build
cd build
cmake --build .
```

Examiner les tests, les comprendre et les lancer.

depuis `build/`

```
./monrepertoire/montest
```

depuis `build\`

```
.\monrepertoire\Debug\montest
```

Informations supplémentaires

Ignorer des tests avec le préfixe `DISABLED_` :

```
TEST(IsPrimeTest, DISABLED_Trivial)
```

Les macros `EXPECT_*` et `ASSERT_*` sont similaires : elles indiquent un comportement non voulu.

- On utilise `EXPECT_*` quand on veut continuer les tests malgré un échec parce qu'on veut révéler plus d'erreurs.
- On utilise `ASSERT_*` quand continuer les tests après un échec n'a pas de sens.

Voir `sample3-unittest.cc`.

L'aménagement de test (*fixture*) se fait avec la macro `TEST_F` et en faisant hérité une classe de `testing::Test`.

Voir `sample3-unittest.cc` à nouveau.

Références et sources

Tests unitaires en Python

https://www.pythoniste.fr/python/linstruction-dassertion-en-python-assert/

https://docs.python.org/3/library/unittest.html (en anglais)

https://docs.python.org/fr/3/library/unittest.html (en français)

https://www.dataquest.io/blog/unit-tests-python/

https://openclassrooms.com/fr/courses/7155841-testez-votre-projet-python

Tests unitaires en Java

https://www.jmdoudoux.fr/java/dej/chap-junit.htm

Couverture en Python

https://coverage.readthedocs.io

Google Test

http://google.github.io/googletest/

Cours de Java de Gérald Oster