Atelier T6.A5
Bien écrire les tests de son composant logiciel Java

Fabrice Ambert *Alexandre Vernotte*

Ecole Polytechnique

6 septembre 2013

Département Informatique des Systèmes Complexes

Test Unitaire

Définition

En programmation informatique, le test unitaire est un procédé permettant de s'assurer du fonctionnement correct d'une partie déterminée d'un logiciel ou d'une portion d'un programme (Wikipédia)

Place dans le cycle de développement en V

La phase de test unitaire prend place immédiatement après la phase de développement.

Acteurs du test unitaire

Qui fait quoi ?

De part sa nature et sa proximité du code source, le test unitaire est pleinement associé à l'activité de développement.

Le test unitaire est à la charge du développeur

Outils du Test Unitaire

Ecrire les tests

- Disponible dans de nombreux langages
- Fonction du langage de programmation
  - Java : JUnit, TestNG
  - PHP : atoum, PHPUnit, SimpleTest

Couverture des tests

Outils d'édition de rapports de couverture du code lors de l'exécution des tests

- Java : Emma, Jacoco, Cobertura
- PHP : Xdebug

Nature des tests

Tests structurels

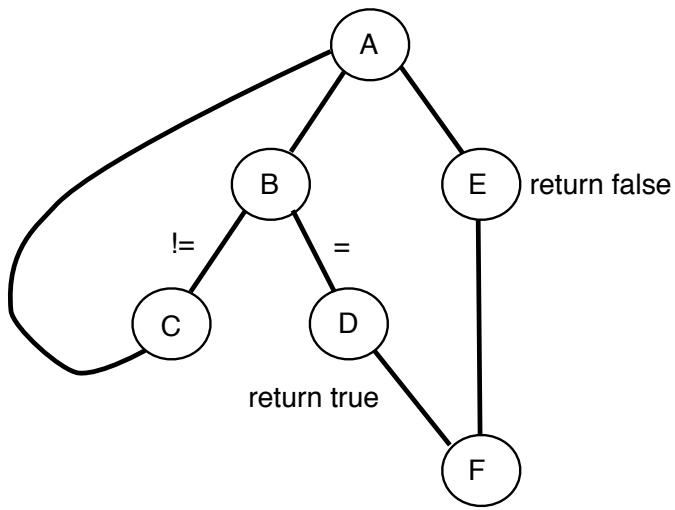
Les Tests Unitaires sont par essence des tests structurels. On teste des portions de programme (boite blanche) pour s'assurer de leur bon fonctionnement.

Quelle couverture viser ?

- Choisir en fonction du code de la méthode et en fonction de la criticité.
- N'importe laquelle des couvertures du flot de contrôle ou du flot de données.
- Souvent parmi "tous noeuds", "tous arcs"... quelque fois chemins

Exemple en Java

```
public <T> boolean
search(List<T> bag, T elt) {
    for (T t : bag) {
        if (t.equals(elt))
            return true;
    }
    return false;
}
```



Exemple en Java

Couvrir tous les arcs

```
@Test
public void testSearch(){
Assert.assertFalse(search(Arrays.asList(1,2,4,736,56,8), 10));
Assert.assertTrue(search(Arrays.asList(1,2,4,736,56,8), 56));
}
```

Couvrir tous chemins indépendants

```
Assert.assertFalse(search(Collections.<Integer>emptyList(), 15));
```

Place des tests dans le projet

Packages et classes de test

A chaque package/classe du source de l'application testée correspond un package/classe de test. En pratique, la classe de test est mise dans le même package que la classe sous test. Le nom de la classe de test rappelle le nom de la classe sous test.

Structure du projet

- Un répertoire pour les sources, un répertoire pour les tests.
- Dans le cas d'un projet maven :

  src/main/java
  src/test/java

JUnit

http ://www.junit.org

Bibliothèque de classes permettant l'expression et l'exécution de tests unitaires

JUnit 4.x

@Test

Balise d'annotation permettant à JUnit de reconnaitre les méthodes de test

org.junit.Assert

Classe de JUnit permettant la vérification des oracles de test

```
@Test
public void testName() {
....
Assert.assertEquals(Oracle, objet.methodeSousTest(...));
}
```

JUnit 4.x

Tester l'apparition d'une exception

```
@Test (expected = ExceptionAttendue.class)
public void testName() throws ExceptionAttendue{
....//mise en état pour production d'exception
objet.methodeSousTestProduisantException(...);
}
```

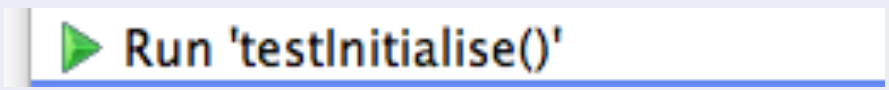
JUnit : exécution des tests

En ligne de commande

java org.junit.runner.JUnitCore TestClass1 [...other test classes...]

résultat dans un fichier html

Depuis un IDE



résultat directement dans IDE

Par une commande maven

mvn test

résultat dans des fichiers xml

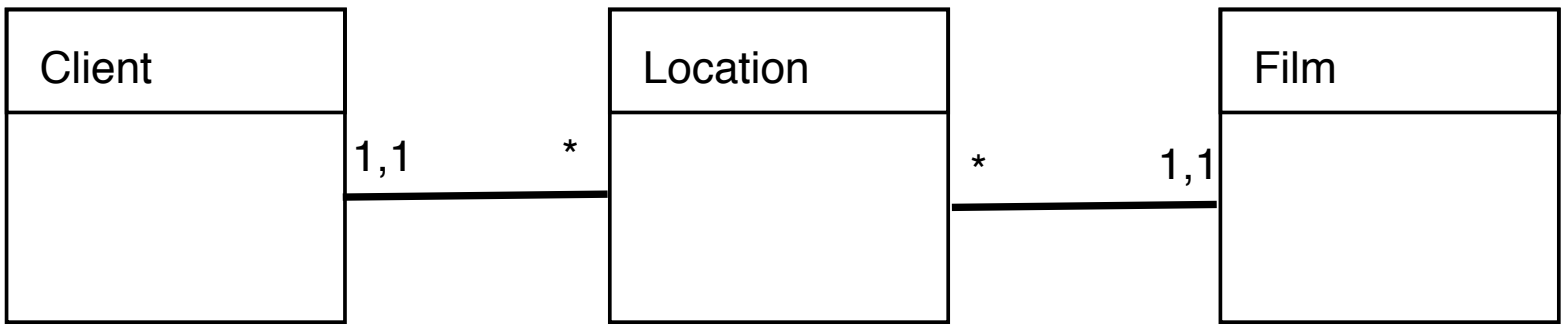
Indépendance des tests

Le résultat d'un test ne doit pas dépendre de l'exécution des tests précédents

Quelques conseils

- Eviter les attributs dans les classes de test
- Ou utiliser de @Before et @After

Test en isolation



Test en isolation

```
public class Location {
    private Film film;
    private Client client;
    ...
    public float montant(int duree) {
        if (client.getCat() == PRIVILEGE)
            return film.prixJour()*(duree-1);
        else ...
```

```
public class Film {
    private Categorie categorie;
    private String titre;
    ...
    public float prixJour() {
        switch (categorie) {
        case Categorie.NOUVEAUTE :
            return categorie.prixBase() * ...;
            ...
```

Test en isolation

Les mocks - objets factices

Les mocks permettent de simuler le comportement d'une classe afin de couper toute dépendance entre classes lors des tests

Mockito - JMocks

http ://code.google.com/p/mockito/

http ://www.jmock.org/

Test en isolation

Utilisation en bouchonnage - stubbing

Le mock retourne les réponses attendues par la classe sous test

```
@Test
public void testMontant() {
    Film film = Mockito.mock(Film.class);
    Mockito.when(film.prixJour()).thenReturn(3.5);
    Client client = Mockito.mock(Client.class);
    Mockito.when(client.getCat()).thenReturn(PRIVILEGE);
    Location loc = new Location(film, client);
    Assert.assertEquals(3.5, loc.montant(2));
}
```

Test en isolation

Utilisation en observateur

Le mock peut observer les appels qui lui sont fait et les mémoriser pour vérification.

```
@Test
public void testMontant() {
    Film film = Mockito.mock(Film.class);
    Mockito.when(film.prixJour()).thenReturn(3.5);
    Client client = Mockito.mock(Client.class);
    Mockito.when(client.getCat()).thenReturn(PRIVILEGE);
    Location loc = new Location(film, client);
    loc.montant(2);
    Mockito.verify(film).prixJour();
}
```

Couverture des tests

Outils

- Emma, Jacoco, Cobertura
- calcule la couverture des lignes de code durant l'exécution des tests
- propose une coloration des lignes couvertes et non couvertes dans un rapport de couverture

Couverture des tests

Utilisation des outils

- Directement depuis un IDE
  - disponible sous forme de plugin de l'IDE
  - coloration directement dans l'IDE
  - pas de publication de rapport
- En ligne de commande

  cobertura-instrument.bat [–basedir dir] [–datafile file] [–destination dir] [–ignore regex] classes [...]
- Via maven

  mvn package -Plocal-coverage

TDD - Test Driven Development

Développement Dirigé par les Tests

- Méthode de développement dans les méthodes agiles
- Préconise l'écriture des tests avant le développement du code

Intérets

- Précise la spécification de la fonctionnalité à implémenter
- Sert d'outil de discussion avec le client
- Constitue une base de tests de non-régression
- Sérénise le développeur dans son activité

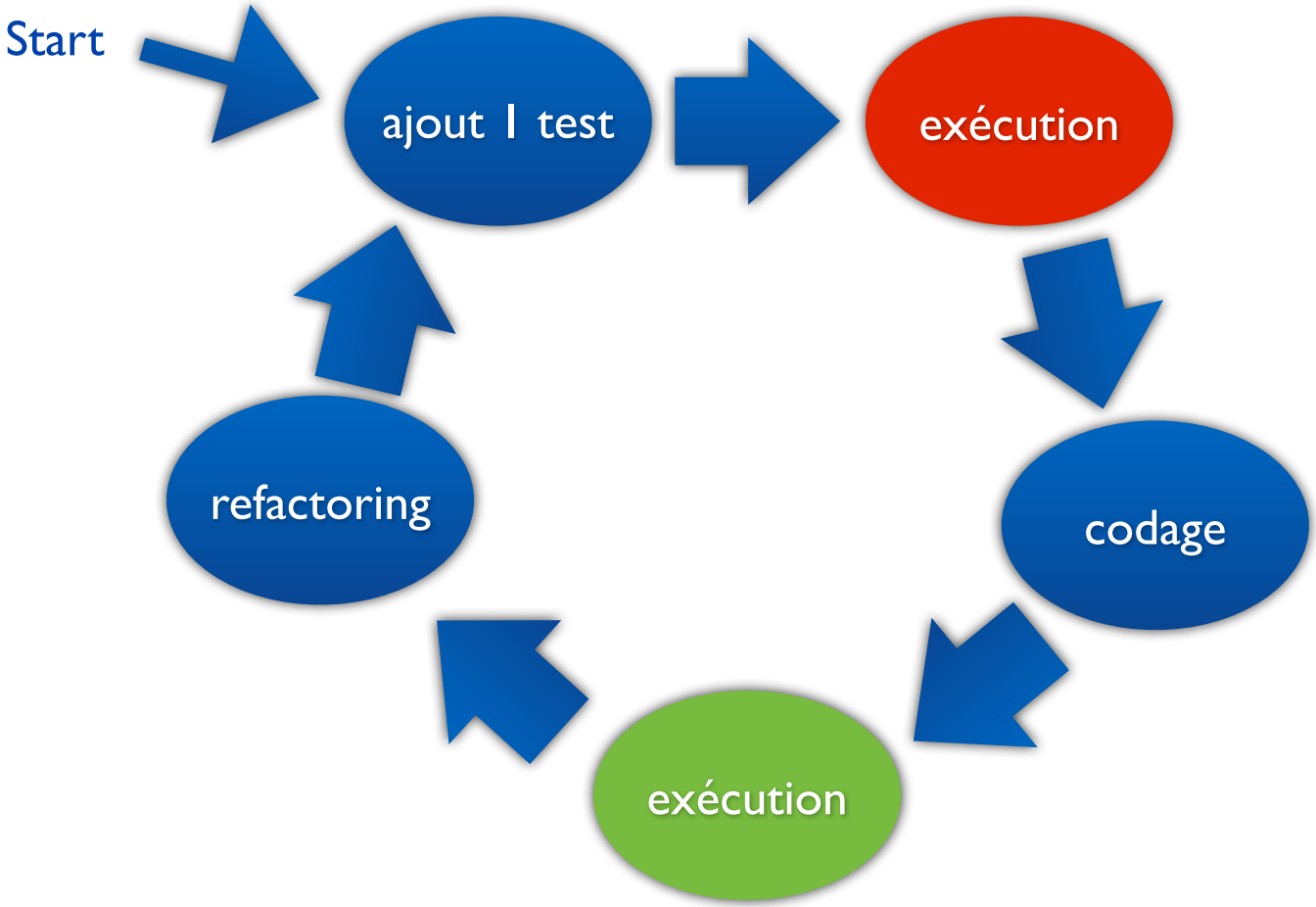
TDD pour le développeur

Le TDD utilise les mêmes outils que le Test Unitaire

TU et TDD

- Les tests écrit dans le cadre du TDD sont des tests unitaires
- Ils confèrent une approche fonctionnelle du TU

Cycle du TDD


Start
ajout 1 test
exécution
codage
exécution
refactoring