



Cabinet d'Architectes en Systèmes d'Information

TDD en pratique (et en Java)

Emmanuel Gaillot & Philippe Kernevez
egaillot@octo.com pkernevez@octo.com

Creative Commons
Contrat Paternité
Pas d'Utilisation Commerciale
Partage des Conditions Initiales à l'Identique
2.0 France



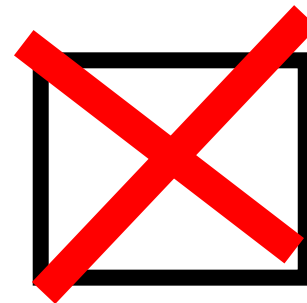
<http://creativecommons.org/licenses/by-nc-sa/2.0/fr/>

Pourquoi faire des tests ?

OK



KO



Quels sont les tests que doit passer un test ?

- Etre **vert** que si le code est valable
 - Anti-pattern: "assertTrue(true);"
 - Risque associé:
 - Investissement à perte
 - Faux sentiment de sécurité
- Etre **rouge** uniquement si le code n'est pas valable
 - Anti-pattern: dépendre de l'état de l'OS, disque, bdd
 - Risque associé:
 - ignorer le résultat du tests (anti pattern "erreur normale")



Cabinet d'Architectes en Systèmes d'Information

Les catégories de test

Tests de recette

- Basés sur la **collaboration** entre le client et l'équipe de **développement**
- Clarifient les **spécifications** en capturant des exemples sous la forme de tests
- Focus sur la **valeur** du logiciel

Tests d'usabilité

- Basés sur la **collaboration entre le client, l'équipe de développement et les ergonomes**
- **Explorer** l'application pour obtenir du feedback sur les fonctionnalités
- Valident **l'usabilité** par le biais de tests manuels permettant d'incarner des *personae*

Tests d'infrastructure

- Basés sur la **collaboration** entre l'équipe de **développement** et les équipes de **production**
- Détectent les **goulots d'étranglement** liés à l'**infrastructure technique**
- **Valident** les performances, la sécurité et la disponibilité de l'application

Tests de développeur

- Basés sur la **collaboration entre développeurs**
- **Améliorent** l'efficacité des développeurs
 - Les tests unitaires apportent de la confiance dans le code et permettent de remanier l'application sans crainte
 - Assurent la **non-régression** en construisant un **harnais** de tests automatiques
- L'équipe écrit des Tests unitaires et **déclare son intention** avant d'écrire du code

Rien de nouveau . . .

JUnit 2

01/08/98 Erich Gamma and Kent Beck

JUnit is a simple framework to write repeatable tests. JUnit can be used with JDK 1.1.* and JDK 1.2.

Et pourtant . . .

- Pourquoi aussi peu de projets sont testés aujourd'hui ?
 - Ca coute trop cher
 - On est déjà à la bourre, on a pas le temps de faire en plus des tests
 - Le build ne passe jamais !
 - On va écrire des tests pour tester le code, mais qui va écrire les tests de nos tests ?
 - Les tests d'Ihm c'est trop compliqué et à la première modifs, il faut tout jeter
- Ca prend trop de temps à s'exécuter (syndrome `-Dmaven.test.skip=true`)
 - De toute façon notre application est trop compliqué, c'est carrément mission impossible

Objectif de la présentation



Cabinet d'Architectes en Systèmes d'Information

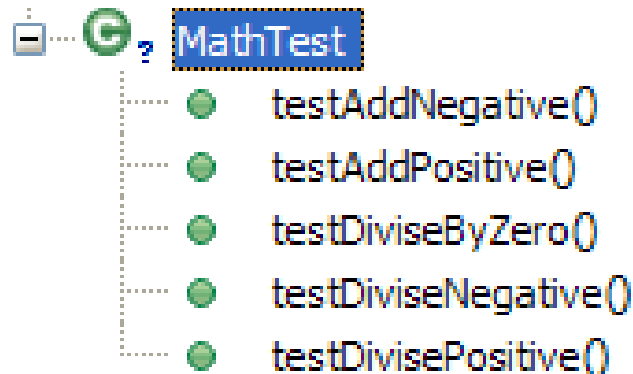
Quelques exemples

Exemple très [trop] simple

- Une classe simple, sans état ni contexte



- Une classe de test qui teste les cas nominaux et les cas aux limites



Exemple: `com.octo.suisse.xpdays.a.toosimple.MathTest`

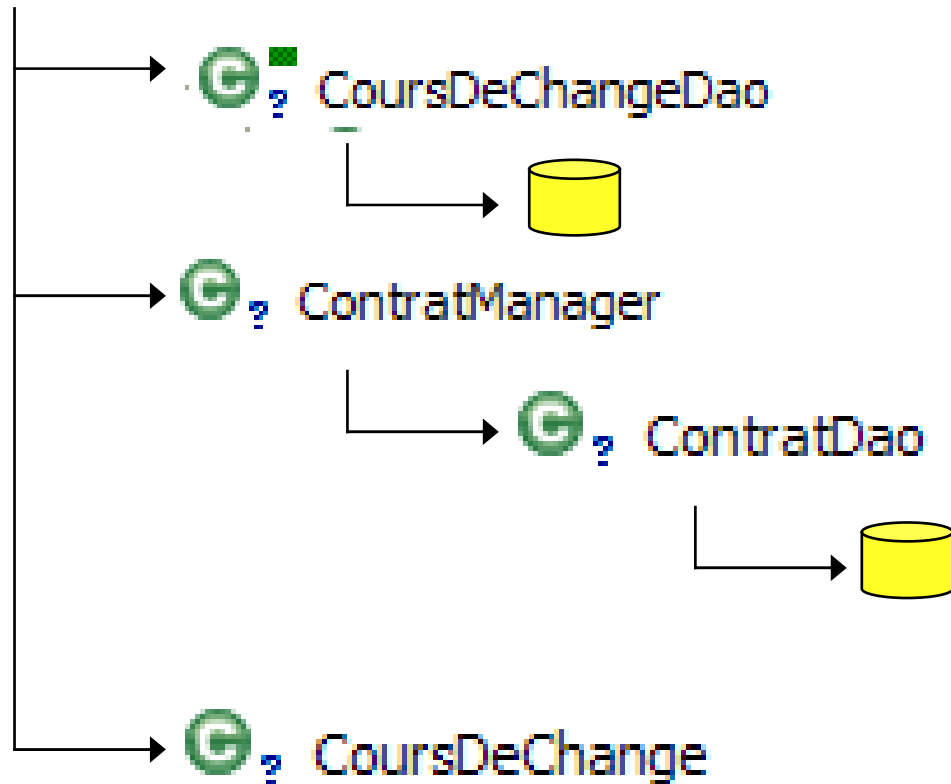
Domaine fonctionnel

- Conversion d'un montant dans une autre devise
 - En recherchant le taux de change à la date données
 - En appliquant les frais du au profil du contrat
 - En fonction du montant du change

*Exemples: com.octo.suisse.xpdays.b.depart.***

Diagramme de séquence

G ? CoursDeChangeManager



Recherche du taux de change

Détermination des commissions

Conversion du montant

*Exemples: com.octo.suisse.xpdays.b.depart.***

Comment tester cet algorithme ?

- En l'état, tester le *legacy* nécessite
 - Un contexte transactionnel
 - Une base de données avec
 - Des contrats
 - Des taux de change
 - Des données en phase avec le code des tests

- Malgré cela les tests ne seront pas très pertinents
 - Ils seront en échec
 - Quand les données seront changées par
 - Un import des données de production
 - Un collègue
 - Quand une des 11 classes introduira une régression

Tester unitairement ?

- Manque de configuration

- Les Managers 'décident' des composants dont ils ont besoins
- Les Dao 'décident' de la session à utiliser
- La factory 'decide' de la configuration bdd à utiliser

➔ Les tests ne peuvent rien configurer, ils doivent utiliser le code de façon monolithique



Cabinet d'Architectes en Systèmes d'Information

Que faire ?

Principe de Separation Of Concern

"Chaque classe doit avoir une responsabilité claire, y compris les classes de test"

- **Bénéfices :**
 - Identification au plus près les défauts du logiciel
 - Possibilité d'intervenir sur le code sans avoir à maîtriser la totalité des couches
 - Exécution plus rapide des tests
- **Exemple de ce qu'il faut éviter:**
 - Chaque couche (re)teste toutes les couches inférieures
 - Tester le container

Comment tester en isolation ?

- 2 patterns
 - Factory configurable
 - Inversion Of Control
- Associés à la notion de Mock (ou bouchon) pour limiter la propagation des appels sur les autres couches
- Remarques:
 - Il n'est pas nécessaire d'utiliser un framework IOC (comme Spring) pour mettre en œuvre le pattern IOC
 - On peut mêler les 2 approches (framework pour la production, à la main pour les tests)

Exemple de mise en œuvre IOC sur les DAO 1/2

Exemples:

com.octo.suisse.xpdays.c.ioc.change.CoursDeChangeDaoTest

- La classe DAO se fait injecter une *SessionFactory*

```
private final SessionFactory mSessionFactory;

@SuppressWarnings("serial")
public class NoCotationException extends Exception
{
}

public CoursDeChangeDao(SessionFactory pSessionFactory)
{
    super();
    mSessionFactory = pSessionFactory;
}

public BigDecimal getTauxDeChange(String pDeviseOrigine, String pDeviseCible, Date pDate)
    throws NoCotationException
{
    Criteria tCriteria = mSessionFactory.getCurrentSession().createCriteria(CoursDeChangeBO.class);
    tCriteria.add(Restrictions.eq("deviseOrigine", pDeviseOrigine));
}
```

Exemple de mise en œuvre IOC sur les DAO 2/2

- Les tests peuvent configurer leur contexte d'exécution
 - Instanciation d'une session factory de test

```
@BeforeClass
public static void initMyClass()
{
    sTestSessionFactory =
        new Configuration().configure("com/octo/suisse/xpdays/c/ioc/Utils/hibernate-configuration-test.xml")
            .buildSessionFactory();
}
```

- Création du jeu de données de test

```
@Before
public void initData()
{
    Session tSession = sTestSessionFactory.getCurrentSession();
    tSession.beginTransaction();
    CoursDeChangeDao tDao = new CoursDeChangeDao(sTestSessionFactory);
    tDao.save(new CoursDeChangeBO("CHF", "EUR", "1.41050", new Date(109, 2, 27, 9, 0, 0)));
}
```

- Exécution du code dans un contexte de test

```
@Test
public void testGetTauxDeChangeNotFound()
{
    try {
        new CoursDeChangeDao(sTestSessionFactory).getTauxDeChange("CHF", "USD", new Date());
        fail("Should not pass");
    } catch (NoCotationException e) { }
}
```

Factory configurable

- Si on peut changer la factory on ajoute un accesseur

```
public class HibernateManager
{
    private static SessionFactory sessionFactory;

    public static void setSessionFactory(SessionFactory pFactory)
    {
        sessionFactory = pFactory;
    }
}
```

- Sinon il est possible d'accéder à un champ/méthode privé via l'introspection

```
Field tField = HibernateManager.class.getDeclaredField("sessionFactory");
tField.setAccessible(true);
tField.set(null, tFact);
```

- *Exemple:*
com.octo.suisse.xpdays.c.ioc.utils.HibernateManagerTest

Tester la couche Service 1/2

- Mise en place d'IOC pour les managers
- Isolation des couches appelées via Mock
 - A la main
 - Création d'une implémentation Mock d'une interface
 - Sous-classage de la vraie classe par un Mock

```
@Test
public void testGetTypeContrat()
{
    ContratManager tManager = new ContratManager(new MockContractDao());
    assertEquals(TypeContrat.GOLD, tManager.getTypeContrat(0));
    assertEquals(TypeContrat.SYLVER, tManager.getTypeContrat(1));
    assertEquals(TypeContrat.BRONZE, tManager.getTypeContrat(2));
}

private static class MockContractDao extends ContratDao
{
    @Override
    public ContratBO read(int pContratId)
    {
        if (pContratId == 0)
        {
            return new ContratBO(0, "G");
        } else if (pContratId == 1)
        {
            return new ContratBO(1, "S");
        } else if (pContratId == 2)
        {
            return new ContratBO(2, "B");
        }
        return super.read(pContratId);
    }
}
```

Tester la couche Service 2/2

- Isolation des couches appelées via Mock

- Avec un framework de Mock (Mockito)

```
@Test
public void testGetTypeContrat()
{
    ContratManager tManager = new ContratManager(getMock());
    assertEquals(TypeContrat.GOLD, tManager.getTypeContrat(0));
    assertEquals(TypeContrat.SYLVER, tManager.getTypeContrat(1));
    assertEquals(TypeContrat.BRONZE, tManager.getTypeContrat(2));
}

private ContratDao getMock()
{
    ContratDao tDao = mock(ContratDao.class);
    when(tDao.read(0)).thenReturn(new ContratBO(0, "G"));
    when(tDao.read(1)).thenReturn(new ContratBO(1, "S"));
    when(tDao.read(2)).thenReturn(new ContratBO(2, "B"));
    return tDao;
}
```

- Exemples:

- com.octo.suisse.xpdays.d.mock.contrat.ContratManagerSousClasseTest
 - com.octo.suisse.xpdays.d.mock.contrat.ContratManagerMockitoTest

Application Web — Servlet 1/2

- Exemple d'une servlet de génération d'image dynamique
- Comment vérifier qu'en
 - Cas d'erreur la servlet renvoie une erreur 404 ?
 - Cas de succès la servlet renvoie 200 et l'image ?
- 2 approches possibles
 - Utilisation de HttpUnit → version Mock
 - Test in container → Cactus

Application Web — Servlet 2/2

```
@Before
public void initClient() throws IOException, SAXException
{
    ServletRunner tSr = new ServletRunner(getWebFile());
    mClient = tSr.newClient();
}

private File getWebFile()
{
    return new File("src/main/webapp/WEB-INF/web.xml");
}

@SuppressWarnings("deprecation")
@Test
public void testDoGetImageNotFound() throws IOException, SAXException
{
    try
    {
        mClient.getResponse("http://localhost/download?Id=Invalid");
        fail("Should not pass !");
    } catch (HttpNotFoundException e)
    {
        assertEquals(404, e.getResponseCode());
    }
}
```

Astuces (interdit -18 ans)...

- Initialisation static qui lève une exception
 - Argh... il faut écraser la classe dans le classpath
- Trop de fichiers Spring (un par test)...
 - Tue Spring (enfer de maintenance)
- Comment s'isoler du code d'un progiciel qui contient la logique métier ?
 - Découplez les responsabilités...
 - Si vous voulez tester le progiciel à cause d'un manque de confiance dans le fournisseur...
 - Faites des tests d'intégration ou fonctionnels
 - Si vous voulez tester votre code
 - Faites des tests en isolation
 - Vous pouvez mêler les 2
 - Mais pas dans le même test !



Référence

