

Tests Réutilisables et Automatisables avec xUnit

Noury Bouraqadi
<http://car.mines-douai.fr/noury>

Option ISIC
Ecole des Mines de Douai

eXtreme Programing & méthodes Agiles

- **eXtreme Programming (XP)**
 - Pousser à l'extrême les bonnes pratiques
- **Méthode "agile" de gestion de projets informatiques**
 - Indépendante du langage (objet ou pas d'ailleurs)
 - Objectifs : Tenir les délais & Garantir la qualité
- **Plus sur**
 - <http://www.extremeprogramming.org>
 - http://fr.wikipedia.org/wiki/Manifeste_agile#Les_12_principes

eXtreme Programing (XP)

- **Quelques Règles :**
 - Scénarios client
 - Identifier les priorités du client
 - Estimation des délais
 - Travail en équipe
 - Programmation en binôme (changeant)
 - Tout le monde partage le code
 - Tests réutilisables et automatisables
 - framework xUnit
 - Intégration continue
 - Jenkins ou autre
 - Itérations planification-codage-release-retrospective
 - 2-3 semaines

Des Tests Réutilisables et Automatiques

- **Exprimer le "cahier des charges"**
 - Vérifier que le contrat est rempli
 - Mesurer l'avancement du projet
 - Ecrire le code le plus simple
- **Qualité croissante**
 - Eviter la répétition d'un bug
 - Permettre l'amélioration "fiable" du code
- **Simplifier l'intégration**
 - Intégration quotidienne
 - Identifier les bugs introduits par l'intégration

SUnit : Framework de test pour Smalltalk

- **Cas de tests** : sous-classes de TestCase
 - **1 test = 1 méthode préfixée de : test**
 - **Méthodes d'instance pour vérifier les invariants**
 - Vérifier les invariant = utiliser des méthodes de Sunit
 - **Ressources pour le test**
 - Initialisation : méthode **setUp**
 - Libération : méthode **tearDown**
- } exécutées à chaque test

Exemple de classe de test

- **TestCase** subclass: #TestSet
instanceVariableNames: 'ensemble' ...
- **setUp**
ensemble := MonEnsemble new.
- **testUnicité**
| element tailleAttendue |
element := Personne nomme: 'Joe'.
tailleAttendue := 1.
ensemble add: element.
ensemble add: element.
self assert: (ensemble includes: element).
self assert: ensemble size **equals:** tailleAttendue

Méthodes pour vérifier les invariants - 1

- **Assertions sur des expressions booléens**
 - assert: expressionBooléenne
 - deny: expressionBooléenne
- **Assertions d'égalité**
 - assert: expression1 equals: expression2
 - assertCollection: collection1 equals: collection2

Méthodes pour vérifier les invariants - 2

- **Assertions sur le temps d'exécution**
 - should: block notTakeMoreThanMilliseconds: anInteger
- **Assertions sur les exceptions**
 - should: bloc raise: classeException
 - shouldnt: bloc raise: classeException

Démarche de développement - 1

1. Partir des spécifications

- Exemple : Un ensemble avec 4 propriétés
 - Un élément ajouté appartient à l'ensemble
 - Un élément ajouté plusieurs fois apparaît une seule fois
 - Un élément supprimé n'appartient plus à l'ensemble
 - Pas d'erreur si suppression d'un élément inexistant

2. Créer une sous-classe de TestCase

- Un champ = ensemble
- la méthode d'initialisation (setUp) : crée l'ensemble
- 4 méthodes de test

Démarche de développement - 2

3. Créer la classe demandée

- Créer la classe *MonEnsemble*

4. Exécutez les tests

5. Déboguez

- Modifiez votre code
- Relancer les tests
- Recommencez jusqu'au succès de tous les tests

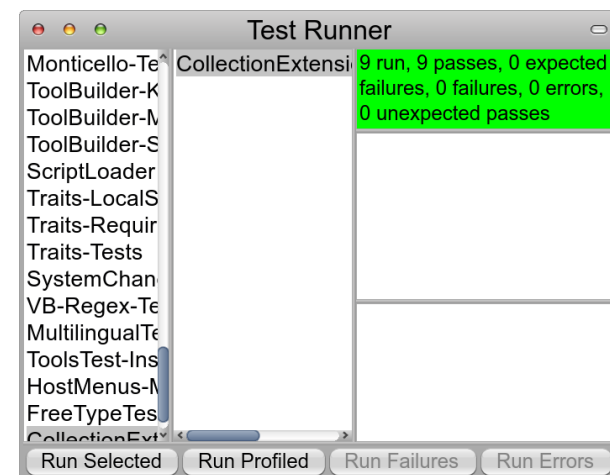
Démarche de développement - 3

• Cycle en spirale

• Itérations :

1. Créer une méthode de test
2. Exécuter les tests
3. Modifier la classe
4. Répéter les étapes 2 et 3 jusqu'à réussir tous les tests

Outil d'exécution de test : TestRunner - 1




Outil d'exécution de test : TestRunner - 2

- **Lance les tests définis dans**
 - 1 cas de test (1 classe dans une sélection)
 - 1 ensemble de tests (plusieurs classes sélectionnées)
- **Résultat = couleur**
 - Vert = OK
 - Jaune = Echec des tests
 - Liste des échecs
 - Possibilité de déboguer
 - Rouge = Erreur non prévue
 - Typiquement méthode non-implémentée ou erreur dans setUp/tearDown
 - Liste des erreurs
 - Possibilité de déboguer

Place de la conception ?

- **Développement 3 temps**
 1. Make it run
 - Code minimal pour faire réussir les tests
 2. Make it right
 - Réviser la **conception**
 - Vérifier que les tests sont toujours verts
 3. Make it fast
 - Optimisation si nécessaire



Cf. Cours 3A