Les Tests Unitaires

Serge Koudoro

Centre de Morphologie Mathématique
Ecole des Mines de Paris

19 juin 2008

Sommaire

Motivation

Débuggage d'un code

- 50% du temps d'un expert
- 90% du temps d'un débutant !
- Réduire ce temps, et augmenter le temps de mise au point d'algorithme

Problème de reprise du code

- Comment fonctionne cet algorithme?
- Fonctionne-t-il toujours ?
- Qu'est ce qui fonctionne ou ne fonctionne pas?
- ...
- Perte de temps, parfois considérable...

Bref rappel de l'état actuel

Morph-M

- 700 tests dans le coeur
- Ensemble des modules Addons testés
- Vérifications automatiques et Régulières
- Evite les Régressions

Bref rappel de l'état actuel

Morph-M

- 700 tests dans le coeur
- Ensemble des modules Addons testés
- Vérifications automatiques et Régulières
- Evite les Régressions

Problème: Module Users

- Mauvaise habitude de ne pas mettre en place une plateforme de test
- Très peu d'algorithmes testés Régulièrement(Python ou c++)
- Difficulté de replonger dans un code d'ancien, aucune assurance qualité...

Sommaire

Qu'est ce qu'un Test Unitaire?

Définition

Bout de code, qui en provoque l'execution d'un autre et qui en analyse le résultat.

Caractéristiques

- Automatique
- Répétable
- Disponible

Principaux Concepts

Quand les mettre en place?

- Avant de commencer à coder afin de donner les grandes lignes directrices
- En meme temps que le développement.
- Légere retouche après !

Principaux Concepts

Quand les mettre en place?

- Avant de commencer à coder afin de donner les grandes lignes directrices
- En meme temps que le développement.
- Légere retouche après !

Documentation

- Exemple d'utilisation de chacune des fonctions de l'algorithmes
- Documentation technique pour chaque algorithmes avec doxygen
- Conclusion: Grande valeur documentaire pour une réutilisation facile !

Principaux Concepts

Robutesse du code

- mettre son application à l'épreuve
- Compte rendu exhaustif de l'application, chose importante avant de la livrer !
- Eviter les tests manuelles incomplets et gourmand en temps.

Principaux Concepts

Robutesse du code

- mettre son application à l'épreuve
- Compte rendu exhaustif de l'application, chose importante avant de la livrer !
- Eviter les tests manuelles incomplets et gourmand en temps.

Non Regression

- Je ne comprends pas pourquoi, hier ca marchait!
- Petite perte de temps à ecrire des tests mais ENORME gain en temps de Débuggage !
- Plus un bug est détecté tot, moins il faudra d'énergie pour le corriger

Quels sont les outils?

Plusieurs framework suivant le language

- java : Junit
- c++ :Cunit,cppUnit,...
- Delphi Dunit
- .Net : Nunit
- ...

Morph-M utilise...

- c++: Boost TestUnit
- python: PyUnit

Sommaire

Framework des Tests Unitaires

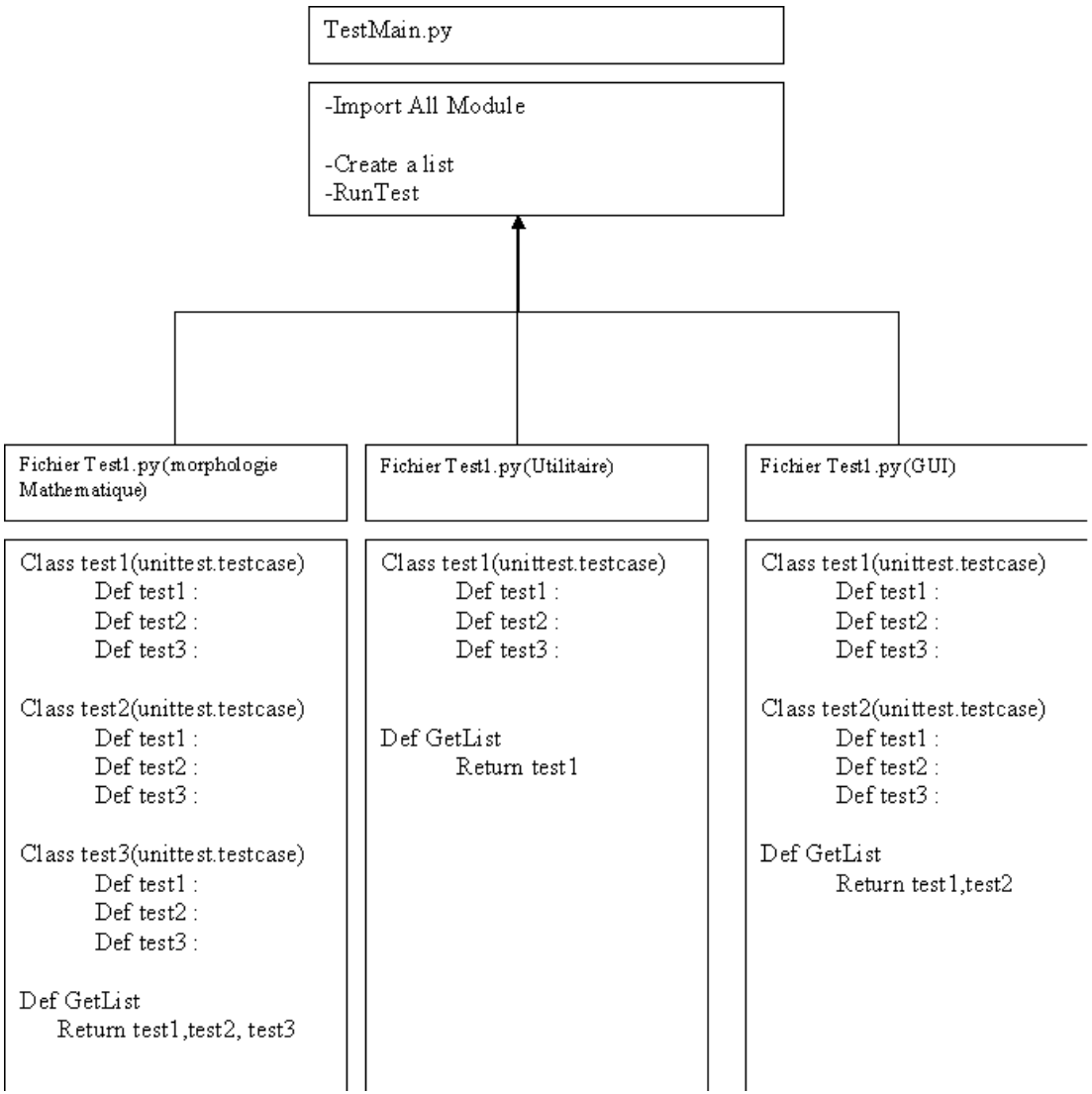
Python : Pyunit Class

1. class TestCase
   - SetUp,tearDown, run,debug...
   - Assert,assert(Not)Equal
   - assertAlmostEqual,...
   - AssertRaises,...
2. class TestSuite
   - addTest,addTests
   - run,debug,countTestCases
3. class TextTestRunner
4. class TestLoader
5. class TestResult
6. ...

C++ : Boost TestUnit Class

1. class TestCase
   - BOOST CHECK, BOOST REQUIRE
   - ...
2. class TestSuite
   - Add
   - BOOST TEST SUITE(name), ...
3. class TestResult
4. class TestLog
5. ...

Architecture Globale



Squellette d'un test

Python

```
#Documentation de la classe
#+ de détails
class TestBasic(unittest.Testcase):
   #Documentation de la fonction à Détailler
   def setUp(self):
        pass
   #Documentation de la function à Détailler
   def tearDown(self):
        pass
   #Documentation de la function
   #@param1 ......
   #@param2 ......
   #@return ....
   def testFunctions1(self):
        pass

class TestBasic1(unittest.Testcase):
   def testFunctions2(self):
        pass
   def testFunctions3(self):
        pass

def RunTests(verbosity=0):
   unittest.main() -->solution 1

   listTEST = ['TestBasic','TestBasic1']
   suiteAll = unittest.TestSuite(listTEST)
   runner = unitTest.TextTestRunner(option)
   runner.run(suiteALL)
```

C++

```
extern testSuite* globalTest

/*! Documentation de la classe
 *@Brief Ma fonction en qq mot
 *
 * + de Détails
*/
class TestBasic()
   void Addtest(){
     globalTest->add(BOOST TEST CASE(&testMaFonction)
     globalTest->add(BOOST TEST CASE(&testMaFonction1
     globalTest->add(BOOST TEST CASE(&testMaFonction2
     }
   /*! Documentation de ma fonction
    *@brief breve explication de la fonction
    *
    *+details
    */
   void TestMaFonction(){
     }
   void TestMaFonction1(){
     }
   void TestMaFonction2(){
     }

void testBasic1(){
  TestBasic::AddTest()
}
```

Exemple: Concretement avec Morph-M...

Python

```
class TestModuleImage(unittest.Testcase):
  def setUp(self):
     self.im = morphee.createImage(morphee.datacategory.dtscalar,morphee.scalrDataType.stdUInt32)
     self.im.setSize(3,3)
     self.im.allocateImage()
  def tearDown(self):
     pass
  def testCreate(self):
     im = morphee.ImCreate(3,5,7,'INT16')
     self.assert(im.isAllocated())
     self.assertEquals( im.getXSize(),3)
     self.assertEquals( im.getDataCategory(),morphee.dataCategory.dtScalar)
     im = morphee.ImCreate(3,5,7,'pixel3<INT16>')
     self.assert(im.isAllocated())
     self.assertEquals( im.getXSize(),3)
     self.assertEquals( im.getDataCategory(),morphee.dataCategory.dtScalar)

  def testCreateSame(self):
      ....

unittest.main()
```

Attention

- Ne pas oublier la documentation
- import unittest
- Fichier séparé!

Exemple: Concretement avec Morph-M...

C++

```
#include <boost/test/unitTest.hpp>
using namespace boost::unitTestFramework;

extern testSuite* imageTest

Class TestClassFunctions
  Public:
    void addtest(){
      imageTest->add(BoostCLASSTestCase(&testFunctions1,instance);
      imageTest->add(BoostCLASSTestCase(&testFunctions2,instance);
      imageTest->add(BoostCLASSTestCase(&testFunctions3,instance);
    }

    void testFunction1(){
      BOOSTMESSAGE('Functions1');BOOSTCHECKPOINT('TestClassFunctions Functions1')
      BOOSTCHECK('Une fonction de mon algo' == RESOK);
      res = ma fonction()
      BOOSTREQUIRE(res = ResOK)
    }

    ....

void ListTest(){
  TestClassFunction::addtest()
}
```

Mauvaises Pratiques

- Eviter de mettre dans le même dossier/fichier les tests et l'application
- Test Triviaux = Perte de temps
- Eviter les effets de Bords ! Tests executable 100, 200, X fois (pendant et Après)
- Ne pas donner d'ordre

Mauvaises Pratiques

- Eviter de mettre dans le même dossier/fichier les tests et l'application
- Test Triviaux = Perte de temps
- Eviter les effets de Bords ! Tests executable 100, 200, X fois (pendant et Après)
- Ne pas donner d'ordre

Bonnes Pratiques

- Une Classe, Une fonction (voir une variable) = Un Test
- Regrouper les tests suivant leurs caractéristiques...
- Tester la Réussite, L'echec, et la cohérence.
- Mettre l'application en difficulté
- Documenter !

Sommaire

Avantages

- Moins compliqué que prévu !!! développement en parallèle...
- Gain de temps !!! Notre métier n'est pas le debuggage...
- Assure une reprise facile du code par un tiers grace à
  - Certitude que les gros bugs Triviaux seront détectés
  - Non regréssion du code
  - Documentation Efficace
- Assure l'efficacité du code,...

Inconvénients

- Tests pertinents viennent avec la pratique
- ??? Aucune raison pour ne pas en Ecrire...

Merci de votre Attention

Ensemble des informations centralisé sur le portail de Morph-M:
`http://rhodes.ensmp.fr/morphee`

Informations Complémentaires:

- Voir le Tutorial et/ou La documentation Développeur
- TP en cours de Réalisation...