Projet de développement

tests unitaires en C, PHP, Python

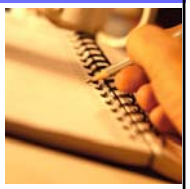
Philippe Collet

Licence 3 Informatique

2009-2010

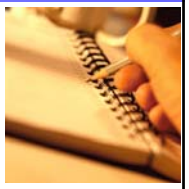
Université Nice SOPHIA ANTIPOLIS

Organisation



- **Cours 1 : principes généraux - svn**
- **Cours 2 : Redmine et gestion de projet**
- **Cours 3 : Introduction à Eclipse**
- **Cours 4 : Eclipse C / PHP, V&V et tests unitaires en Java**
- **Cours 5 : Tests C / PHP / Python**
- **Cours 6 : compléments, documentation , soutenance**

Plan



- **Infrastructure Check**
- **SimpleTest**
- **PyUnit**
- **Application à vos projets**

L'infrastructure Check

Université Nice SOPHIA ANTIPOLIS

Check (check.sourceforge.net)

- **Environnement de tests unitaires pour le langage C**
  - Inspiré de JUnit (pour Java)
  - Plus ou moins intégré à l'approche *Extreme Programming*
- **D'autres outils similaires : GNU AutoUnit, cUnit…**
- **Surtout, beaucoup d'infrastructures dédiées au C++ :**
  - cppUnit,
  - cppTest,
  - unitTest++,
  - Boost.Test,
  - CppUnitLite,
  - NanoCppUnit,
  - Unit++,
  - CxxTest,
  - etc.

Principe de fonctionnement

- **En Java, facile…**
  - Un échec lève une exception et hop !
- **En C… on risque plutôt un crash dans l'espace d'adressage !**
- **Check**
  - utilise fork pour créer un nouvel espace d'adressage pour chaque test unitaire
  - utilise des files de messages pour retourner des comptes-rendus à l'environnement de tests
- **Pour piloter le lancement des tests, check**
  - Peut être appelée directement, mais surtout
  - Depuis un makefile, et encore mieux
  - Depuis autoconf/automake pour générer le tout…

Écrire un test

- **Utiliser l'include** `#include <check.h>`
- **Ecrire le test entre les deux macros**

```
START_TEST (test_name)
{
   /* unit test code */
}
END_TEST
```


Exemple

```
START_TEST (test_create)
{
 Money *m;
 m = money_create(5, "USD");
 fail_unless(money_amount(m) == 5,
             "Amount not correct on creation");
 fail_unless(strcmp(money_currency(m), "USD") != 0,
             "Currency not set correctly on creation");
 money_free(m);
}
END_TEST
```

ORACLE

Alternatives pour les oracles

```
if (strcmp(money_currency(m), "USD") != 0) {
  fail("Currency not set correctly on creation");
}
```

Aussi équivalent à

```
fail_if(strcmp(money_currency(m), "USD") != 0,
        "Currency not set correctly on creation");
```

```
fail_unless(money_amount(m) == 5, NULL);
```

Est équivalent à

```
fail_unless(money_amount(m) == 5,
            "Assertion 'money_amount (m) == 5' failed");
```

Liste variable d'arguments, style *printf* :

```
fail_unless(money_amount(m) == 5,
            "Amount was %d, instead of 5", money_amount(m));
```

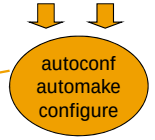

Pilotage depuis automake/autoconf

Makefile.am

```
TESTS=check_money
noinst_PROGRAMS=check_money
check_money_SOURCES= money.h money.c check_money.c
check_money_INCLUDES= @CHECK_CFLAGS@
check_money_LIBS= @CHECK_LIBS@
```

configure.in (vue partielle)

```
AC_INIT()
AM_INIT_AUTOMAKE()
AC_PROG_CC
AM_PATH_CHECK()
AC_OUTPUT(Makefile)
```



gentest

```
aclocal
autoconf
autoheader
automake --add-missing
./configure
```

Makefile

```
matrix> make -k check
```


Exemple complet : le retour de Money

- **Money.h**

```
typedef struct Money Money;
Money *money_create(int amount, char *currency);
int money_amount(Money *m);
char *money_currency(Money *m);
void money_free(Money *m);
```

- **Money.c**

```
#include <stdlib.h>
#include "money.h"
Money *money_create(int amount, char *currency) {
   return NULL; }
int money_amount(Money *m) { return 0; }
char *money_currency(Money *m) { return NULL; }
void money_free(Money *m) { return; }
```


Jeu de tests

- **Création des cas de test**
- **Regroupement dans une *suite***
  - *Des test cases*
  - Pour chacun, des tests sont ajoutés
- **Exécution depuis un *suite runner***
  - À partir d'une *suite*
  - Exécution en mode normal
  - Récupération du retour (qui sera interprété par make)

```
#include <stdlib.h>
#include <check.h>
#include "money.h"

START_TEST (test_create)
…
END_TEST

Suite *money_suite(void) {
  Suite *s = suite_create("Money");
  TCase *tc_core = tcase_create("Core");
  suite_add_tcase (s, tc_core);
  tcase_add_test(tc_core, test_create);
  return s;
}

int main(void) {
  int nf;
  Suite *s = money_suite();
  SRunner *sr = srunner_create(s);
  srunner_run_all(sr, CK_NORMAL);
  nf = srunner_ntests_failed(sr);
  srunner_free(sr);
  return (nf == 0) ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

Paramétrage du *SRunner*

`void srunner_run_all(SRunner *sr, enum print_output print_mode);`

- Exécute tous les tests unitaires de tous les cas de test définis pour toutes les suites dans `sr`
- Collecte les résultats dans `sr`
- Affiche les résultats selon le `print_mode`
  - CK_SILENT : aucune sortie
  - CK_MINIMAL : résumé (number run, passed, failed, errors).
  - CK_NORMAL : résumé + un message par test échoué
  - CK_VERBOSE : résumé + un message par test (passé, échoué)
  - CK_ENV : récupère le mode depuis la variable d'environnement CK_VERBOSITY ("silent", "minimal", "normal, "verbose«)

❒ **Pour les SRunners déjà exécutés, la fonction d'affichage séparée suivante peut être utilisée :**

```
void srunner_print(SRunner *sr, enum print_output print_mode);
```


Passons les tests

❒ **En mode CK_NORMAL**

```
matrix> make –k check
...
...
Running suite(s): Money
0%: Checks: 1, Failures: 1, Errors: 0
check_money.c:20:F:Core:test_create: Amount not set correctly
on creation
FAIL: check_money
==================
1 of 1 tests failed
==================
make[1]: *** [check-TESTS] Error 1
make[1]: Leaving directory `/home/collet/checkTestMoney'
make: *** [check-am] Error 2
make: Target `check' not remade because of errors.
```


Test fixtures

❒ **Plusieurs tests peuvent utiliser les mêmes données**

- Un code d'initialisation constant pour tous les tests concernés
- Test fixture = code *setup* et/ou *teardown*

❒ ***Checked fixtures***

- Exécutés dans l'espace d'adressage du test unitaire
- Avant chaque test unitaire dans un Tcase, la fonction *setup* est exécutée
- Après chaque test unitaire dans un Tcase, la fonction *teardown* est exécutée
- Si le code renvoie des signaux ou échoue, ce sera rattrapé et reporté par le *SRunner*

❒ ***Unchecked fixtures***

- Exécutés dans l'espace d'adressage du programme de test
- Pas de signaux, pas de sortie, mais utilisation possible de fonctions *fail*
- *Setup* et *teardown* sont resp. exécutées avant/après le cas de test

Exemples de *Test Fixture*

1. **Définir les variables globales et les fonctions de signature void (void)**

```
Money *five_dollars;
void setup(void) {
  five_dollars = money_create(5, "USD");
}
void teardown(void) {
  money_free(five_dollars);
}
```

2. **Ajouter les fonctions au TCase avec** *tcase_add_checked_fixture*

```
tcase_add_checked_fixture(tc_core, setup, teardown);
```

3. **Écriture du test en conséquence (réécriture de notre 1er test)**

```
START_TEST (test_create) {
   fail_unless(money_amount(five_dollars) == 5,
               "Amount not set correctly on creation");
   fail_unless (strcmp(money_currency(five_dollars), "USD") == 0,
                "Currency not set correctly on creation");
} END_TEST
```

Check : installation et utilisation

- **Récupérer l'archive (http://check.sourceforge.net/)**
  - Installer les bibliothèques (si vous avez les droits)
  - Ou la référencer dans un makefile

```
SOURCES= checkFile0.c file0.c
EXECS = checkFile0

LDFLAGS = -L/usr/lib
CFLAGS  = -I/usr/include

LIBS = -lcheck

all: test

test: $(EXECS)
	$(EXECS)

$(EXECS): $(SOURCES)
	gcc $(CFLAGS) $^ $(LDFLAGS) $(LIBS) -o $@

clean:
	rm -f *.o *~ \#* $(EXECS) *.exe

file0.c: file.h
```

- **Plus d'info :**
  - http://check.sourceforge.net/doc/check.html/index.html

SimpleTest

SimpleTest

- **Framework de test unitaire pour PHP (4 et 5)**
  - Créé par Marcus Baker
- **Fonctionnalités**
  - Principes similaires à JUnit (et PhpUnit)
  - Test passé coté client ou déployé sur le serveur
  - Support pour des « objets fantaisie » (*mock objects*)
  - Possibilité d'automatisation de test de non-régression d'applications web
    - Client HTTP « scriptable » qui analyse des pages web, simule des actions comme des clics sur les liens et soumet des formulaires
- **Alternatives :**
  - PhpUnit

SimpleTest : fonctionnement de base (extrait du tutorial)

- **Nous disposons d'une classe *classes/Log.php* qui fait… du log**
  - C'est la classe à tester
- **Etape 1 : script de test (*tests/log_test.php*)**

```
<?
php require_once('simpletest/autorun.php');
require_once('../classes/log.php');
class TestOfLogging extends UnitTestCase {
}
?>
```

- **Répertoire *simpletest* (dépendant de l'installation de simpleTest)**
  - dans le dossier courant
  - ou dans les dossiers pour fichiers inclus.
- **Fichier "autorun.php"**
  - inclut les éléments de SimpleTest
  - lance aussi les tests pour nous

SimpleTest : fonctionnement de base

- **Supposons que la classe Log prenne le nom du fichier à écrire au sein du constructeur, et que nous avons un répertoire temporaire dans lequel placer ce fichier**

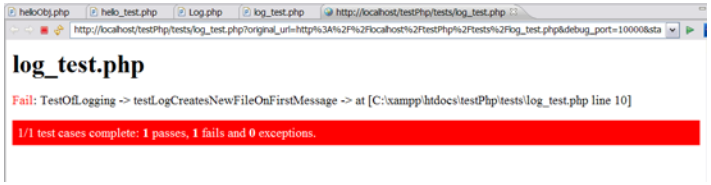
```
<?
php require_once('simpletest/autorun.php');
require_once('../classes/log.php');
class TestOfLogging extends UnitTestCase {
    function testLogCreatesNewFileOnFirstMessage() {
        @unlink('/temp/test.log');
        $log = new Log('/temp/test.log');
        $this->assertFalse(file_exists('/temp/test.log'));
        $log->message('Should write this to a file');
        $this->assertTrue(file_exists('/temp/test.log'));
    }
}
?>
```


SimpleTest : exécution

- **Au lancement du scénario de test :**
  - **Identification des méthodes commençant par « test »**
  - **En cas d'erreur ou d'échec, le framework affiche tout de suite le résultat**
- **Comment lancer ?**
  - **Afficher la page dans le navigateur**
  - **En local (si on a un interprète sur sa machine : station de travail)**
  - **A distance (avec l'environnement www-mips de la faculté)**

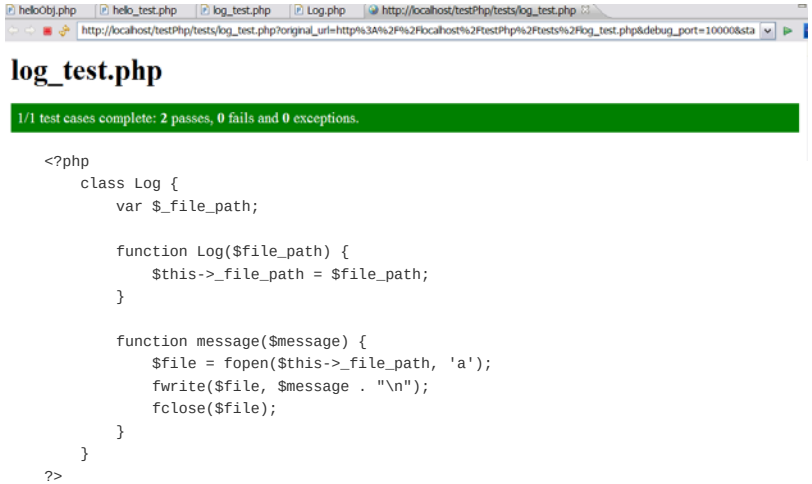
SimpleTest : résultats

- **A surveiller :**
  - Bonne inclusion (locale) des fichiers testés
  - Bonne inclusion des fichiers de simpleTest : prendre le répertoire de simpletest dans le plugin Eclipse pour le recopier dans votre serveur de test (cf. « include_path » de votre php.ini, par exemple)



- **= Echec**
- **Sinon, sortie « verte » pour réussite des tests**

SimpleTest : tests passés



Suite de tests

- **Nouveau fichier : tests/all_tests.php**

```
<?php
require_once('simpletest/autorun.php');

class AllTests extends TestSuite {
    function AllTests() {
        $this->TestSuite('All tests');
        $this->addFile('log_test.php');
    }
}
?>
```

- **Autorun.php permet l'auto-exécution du script**
- **TestSuite doit chaîner son constructeur (limitation de simpletest)**

Développement de l'exemple

- **Supposons une classe SessionPool qui utilise le Log…**

```
class SessionPool {
    ...
    function logIn($username) {
        ...
        $this->_log->message('User $username logged in.');
        ...
    }
    ...
}
```


Setup et Teardown (à la JUnit)

```
<?php
require_once('simpletest/autorun.php');
require_once('../classes/log.php');
require_once('../classes/session_pool.php');

class TestOfSessionLogging extends UnitTestCase {

    function setUp() {
        @unlink('/temp/test.log');
    }

    function tearDown() {
        @unlink('/temp/test.log');
    }

    function testLoggingInIsLogged() {
        $log = new Log('/temp/test.log');
        $session_pool = &new SessionPool($log);
        $session_pool->logIn('fred');
        $messages = file('/temp/test.log');
        $this->assertEqual($messages[0], "User fred logged in.\n");
    }
}
?>
```


Test de page web

- **Possibilité de tester une page web**
  - Assez basique (pas de javascript)
- **Fonctionnalités**
  - Charger une page, naviguer

```
<?php
require_once('simpletest/autorun.php');
require_once('simpletest/web_tester.php');

class TestOfAbout extends WebTestCase {
    function testOurAboutPageGivesFreeReignToOurEgo() {
        $this->get('http://test-server/index.php');
        $this->click('About');
        $this->assertTitle('About why we are so great');
        $this->assertText('We are really great');
    }
}
?>
```

Test de page web

- **Fonctionnalité**
  - Naviguer à travers des formulaires

```
<?php
require_once('simpletest/autorun.php');
require_once('simpletest/web_tester.php');

class TestOfRankings extends WebTestCase {
    function testWeAreTopOfGoogle() {
        $this->get('http://google.com/');
        $this->setField('q', 'simpletest');
        $this->click("I'm Feeling Lucky");
        $this->assertTitle('SimpleTest - Unit Testing for PHP');
    }
}
?>
```


Suite du tutorial

- http://simpletest.org/fr/start-testing.html
- http://simpletest.org/fr/first_test_tutorial.html

PyUnit

Université Nice SOPHIA ANTIPOLIS

Support standard des tests unitaires en Python

- **Aussi appelé PyUnit**
  - **Standard depuis la version 2.1**
- **Fonctionnalités**
  - Principes similaires à Junit **version 3**
  - Test Case, Test Suite, Test Fixture, Test Runner
- **Facile quand on connaît les principes de Junit...**

pyUnit : Les bases du TestCase

- Importer unittest
- Hériter de TestCase dans une classe
- Définir des fonctions test_XXX(self)
- Les implémenter avec des oracles self.assertXXX
  - assertEqual
  - assert_
  - assertRaises : pour les exceptions
- Définir une méthode « fixture » setUp(self)
- Définir si nécessaire une méthode tearDown(self)

TestCase complet avec fixture

```
import random
import unittest

class TestSequenceFunctions(unittest.TestCase):

    def setUp(self):
        self.seq = range(10)

    def test_shuffle(self):
    # make sure the shuffled sequence does not lose any elements
        random.shuffle(self.seq)
        self.seq.sort()
        self.assertEqual(self.seq, range(10))
```


TestCase complet avec fixture (suite)

```
def test_choice(self):
        element = random.choice(self.seq)
        self.assertTrue(element in self.seq)

 def test_sample(self):
     self.assertRaises(ValueError, random.sample, self.seq, 20)
     for element in random.sample(self.seq, 5):
         self.assertTrue(element in self.seq)

if __name__ == '__main__':
    unittest.main()
```


Exécution des tests

- **unittest.main()**
  - Interface en ligne de commande pour exécuter les tests de la classe

```
...
----------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

```
suite = unittest.TestLoader().loadTestsFromTestCase(TestSequenceFunctions)
unittest.TextTestRunner(verbosity=2).run(suite)
```

```
test_choice (__main__.TestSequenceFunctions) ... Ok
test_sample (__main__.TestSequenceFunctions) ... Ok
test_shuffle (__main__.TestSequenceFunctions) ... Ok
----------------------------------------------------------------
Ran 3 tests in 0.110s

OK
```

Application à vos projets

- **Tout le code doit être documenté raisonnablement**
  - **fichiers source en C**
  - **toute classe en Java**
  - **pages et classes PHP**
- **Comment ?**
  - **entête du module/classe**
  - **commentaire sur les fonctions/opérations les plus importantes**
  - **(pas forcément tous les getters/setters)**