Sofeware Engingeering

Unit test 조사

---

4 조
김진욱
이희재
김태준
서지혁

1. Unit test 개념

1) Unit test란?

유닛 테스트(unit test)는 컴퓨터 프로그래밍에서 소스 코드의 특정 모듈이 의도된 대로 정확히 작동하는지 검증하는 절차다. 즉, 모든 함수와 메소드에 대한 테스트 케이스(Test case)를 작성하는 절차를 말한다. 이를 통해서 언제라도 코드 변경으로 인해 문제가 발생할 경우, 단시간 내에 이를 파악하고 바로 잡을 수 있도록 해준다. 이상적으로, 각 테스트 케이스는 서로 분리되어야 한다. 이를 위해 가짜 객체(Mock object)를 생성하는 것도 좋은 방법이다. 유닛 테스트는 (일반적인 테스트와 달리) 개발자(developer)뿐만 아니라 보다 더 심도 있는 테스트를 위해 테스터(tester)에 의해 수행되기도 한다.

2) 수행방법?

1)테스트 준비

① 테스트 계획을 점검하고 각 단위 테스트별 담당자를 결정한다.

- 일반적으로 단위 테스트는 해당 개발자에 의해 수행되나 필요한 경우, 고객이나 다른 프로젝트 멤버들로 구성된 테스트를 위한 별도의 팀을 구성하여 단위 테스트를 수행할 수 있다.

② 단위 테스트를 위한 별도의 환경이 필요한 경우 이를 구성한다.

- 일반적으로 개발 Tool에서 지원하는 테스트 환경을 활용하여 단위 테스트를 수행하나, 단위 테스트를 위한 별도의 환경이 필요한 경우 이를 구성한다.

2) 단위 테스트를 위한 테스트 케이스를 작성한다.

① 각 Class의 모든 메소드(Operation)에 대한 테스트가 가능하도록 테스트 케이스를 작성한다.

② Use Case의 모든 기능에 대한 테스트가 가능하도록 테스트 케이스를 작성한다.

3) 단위 테스트를 위한 별도의 데이터가 필요한 경우 데이터를 준비한다.

4) 담당자를 중심으로 단위 테스트를 실시하고 그 결과를 테스트 케이스에 기술한다.

5) 발견된 오류에 대해 수정 작업을 수행한 후 테스트를 반복해서 수행한다.

6) 테스트 결과를 보고한다.

2. Unit test 도구

1) Check

①설명

Check는 C를 위한 Unit test framework이다. check의 가장 큰 특징으로 볼 수 있는 것은 각 테스트 케이스를 실행할 때 fork() 시스템 콜을 통해 별도의 프로세스를 생성하여 실행한다는 것이다. 예외 처리를 제공하는 다른 언어들과 달리 C 언어에서는 정상적인 실행 이외의 상황 즉, segmentation fault가 발생하거나, 내부에서 exit()를 호출하여 비정상 종료되는 경우 테스트 케이스 실행 도중 프로그램 전체가 종료되어 버릴 수 있다. 이 경우 테스트에 관련된 아무런 결과를 보고받을 수 없으므로 이러한 문제를 극복하기 위해 분리된 별도의 프로세스에서 테스트를 수행하고 부모 프로세스가 그 결과 및 종료 상태를 감지하여 안전하게 테스트 결과를 보고해 준다.

②사용 방법

#include <check.h> 해서 헤더파일을 추가해주고,

START_TEST (test_name)

{

/* unit test code */

}

END_TEST

START_TEST랑 END_TEST사이에 테스트할 코드를 넣어준다.

③실행 화면

```c
#include <stdio.h>
#include <check.h>

START_TEST(test_check)
{
        fail_unless(1 + 1 == 3, "What's wrong? 1 + 1 = %d", 1 + 1);//두번째 인자는 실패시 출력할 메시지이다.
}
END_TEST

START_TEST(test_check_new_api)
{
        ck_assert(1);
          ck_assert_int_eq(1 + 1, 2);
            _ck_assert_int(1, <, 2);
              ck_assert_str_eq("test_check_new_api", __FUNCTION__);
}
END_TEST

Suite *
my_suite(void)
{
        Suite *s = suite_create("My Test Suite");

          TCase *tc = tcase_create("My Test Case");
            tcase_add_test(tc, test_check);
              tcase_add_test(tc, test_check_new_api);

                suite_add_tcase(s, tc);
                  return s;
}

int main (void)
{
        Suite *s = my_suite();
          SRunner *sr = srunner_create(s);
            srunner_run_all(sr, CK_ENV);
              return 0;
}
```

```
User@User-PC ~
$ gcc test.c -lcheck -o test

User@User-PC ~
$ ./test
Running suite(s): My Test Suite
50%: Checks: 2, Failures: 1, Errors: 0
test.c:6:F:My Test Case:test_check:0: What's wrong? 1 + 1 = 2
```

2) C unit

①설명

C unit는 기본 단위 테스트를 위한 도구이다. CUnit 디렉토리 밑에 실제 프레임워크를 담당하는 코드들이 있고 그 중에서 automated.c파일에는 테스트 결과를 xml 파일로 만들어 주는 코드가 주종을 이룬다. 다른 basic, console은 커맨드 창에 결과를 보여준다. Curse는 리눅스 계열에서만 사용 할 수 있는 테스트 결과를 보여준다. 중요한 점은 unit test가 단순하게 생각한다면 black box 테스트라는 점이다. 즉, 입력 값을 주고 원하는 결과값이 나오는지 테스트할 뿐이란 점이다. 그리고 대부분 boundary test case를 사용하는 용도로 쓰일 수 있다는 점이다. Code level test를 원한다면 더욱 더 정교한 unit test 툴이 필요하다.

②사용 방법

1) **테스트 함수 작성**
2) **Regisry에 등록 (CU_initialize_regisry() 사용)**
3) **Regisry에 Suite 등록 (CU_add_suite() 사용)**
4) **Suite에 Test 등록 (CU_add_test() 사용)**
5) **Running모드 함수 사용 (CU_console_run_tests() 등)**
6) **Regisry 초기화 (CU_cleanup_regisry() 사용)**

④실행 화면

```
konkuk@NB48 /home/CUnit-2.1-2
$ ./test


     CUnit - A unit testing framework for C - Version 2.1-2
     http://cunit.sourceforge.net/


Suite: Suite_1
  Test: test of fprintf() ...passed
  Test: test of fread() ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests      2      2      2      0        0
             asserts      5      5      5      0      n/a

Elapsed time =    0.000 seconds
```

3) simple test

①설명

C와 C++을 위한 오버헤드가 적은 간단한 테스팅 도구이다. C나 C++의 경우 다른 프레이워크들과 달리 reflection 기능이 없기에 이 문제를 해결하기 위해 만들었다. (reflection 기능은 해당 프로그램을 조사하거나 스스로 살펴보는 기능으로 프로그램 내부의 property를 조작할 수 있게 함. 해당 프로그램 안에 정의된 함수에 대한 정보를 얻을 수 있는 방법이 reflection). 다른 테스팅 기법과의 차별점은 각각의 테스트를 한번만 작성하면 되며, perl같은 script 언어를 쓰지 않아도 된다는 점이다. 또한 어떤 시스템이나 컴파일러에서든 호환성이 좋다.

②사용 방법

#include "tests.h"로 헤더파일을 추가해 주고,

START_TEST (test_name)

{

  /* unit test code */

}

END_TEST

START_TEST랑 END_TEST사이에 테스트할 코드를 넣어준다.

```
#include "tests.h"

// Start the overall test suite
START_TESTS()

// A new group of tests, with an identifier
START_TEST("simple")
        // We then write the tests we want to check
        ASSERT(1 == 1);
        ASSERT_EQUALS_FLOAT(1, 1);
END_TEST()

START_TEST("fail")
        // These tests will fail, and we will get output.
        ASSERT(1 == 0);
        ASSERT_EQUALS_FLOAT(1, 0);

        // Lets give a description of the test, before it
        // fails - this will be printed out instead.
        TEST("we expect this test to fail. (3==2)");
        ASSERT(3 == 2);
END_TEST()
```

③실행 화면

```
희 재 @heejae ~
$ ./simpletests.exe
> simple...
> fail...
[FAIL] simpletests.c:15 : (fail) : 1 == 0 fails
[FAIL] simpletests.c:16 : (fail) : 1 is supposed to equal (float) 0
       (1.000000 != 0.000000)
[FAIL] simpletests.c:21 : (fail) : we expect this test to fail. (3==2)

--- Results ---
Tests run:    2
Passes:       2
Failures:     3
```