韓國 컴퓨터情報學會 論文誌
第10卷 第1號, 2005. 3.

2005-3-1-1-3

C환경에서의 XP적용을 위한 모크객체생성기에 관한 연구

정 영 목*, 박 제 원**, 이 남 용***

A Study of Mock Objects Generator for Applying XPwith Special Attention to C

Young-Mok Jung*, Jae-Won Park**, Nam-Yong Lee***

요 약

XP(Extreme Programming)는 경량소프트웨어개발방법론 중의 하나로서 의사소통, 단순함, 피드백, 용기의 네 가지 가치추구를 통해 소프트웨어의 생산성과 품질을 향상시키는 실질적인 소프트웨어개발방법론이다. 그러나 XP의 핵심실천사항 중 하나인 테스트주도개발(Test Driven Development)은 한 가지 문제점을 가지고 있다. 테스트수행시간이 오래 걸리거나 테스트케이스(Test Cases)의 독립성을 확보하지 못할 경우 테스트주도개발이 불가능하다는 문제로서 최근 이를 해결하기 위해 모크객체(Mock Objects)의 중요성이 강조되고 있다. 모크객체는 문제가 되는 실제코드를 흉내내는 객체로서 수동으로 작성할 수 있을 만큼 간단함을 원칙으로 하지만, 모크객체를 수동으로 작성하는 것은 실제로는 비효율적인 작업이기 때문에 자바에서는 MockObjects, EasyMock와 같은 모크객체생성기를 사용하고 있다. 그러나 모크객체는 객체지향을 전제로 하고, C언어용 모크객체생성기도 없기 때문에 C언어에서는 모크객체를 적용하기가 곤란하다. 본 논문에서는 객체지향개념이 취약한 C언어에서 모크객체를 손쉽게 생성하기 위한 C언어용 모크객체생성기 CMock를 제시하고 이의 효용성을 검증하는 연구를 수행하였다.

Abstract

As one of the lightweight software development methodology, the XP (Extreme Programming) is the practical means to improve the productivity and quality of software through the pursuit of 4 values - communication, simplicity, feedback, and courage. It appears, however, the TDD (Test Driven Development), one of the practices of XP, has a problem, which is the unavailability of the test driven development in case of the prolonged period of testing or the failure of securing the independency of the test cases. This results in the emphasis on the importance of the Mock Objects recently. The Mock Objects, the one imitating the faulty real code, has the fundamentals of simplicity allowing even manual script but, due to the inefficiency of manual script of the Mock Objects in a real life, it is implemented the Mock Objects Generator such as MockObjects, EasyMock in Java. It is found difficult, however, to apply the Mock Objects in C language due to its object-oriented premise as well as the absence of mock objects generators for C language. Therefore, in this paper, it is presented the CMock, a Mock Objects generator for C language which allows the easy creation of the Mock Objects, and the study is performed to verify the efficiency accordingly.

▸ Keyword : Mock Objects, Test Driven Development, Extreme Programming

---

• 제1저자 : 정영목
• 접수일 : 2005.01.06, 심사완료일 : 2005.02.26
* 숭실대학교 대학원 컴퓨터학과 소프트웨어공학전공 석사과정
** 숭실대학교 대학원 컴퓨터학과 소프트웨어공학전공 석사과정, *** 숭실대학교 컴퓨터학부 교수

Ⅰ. 서 론

XP(Extreme Programming)는 경량소프트웨어개발방법론중의 하나로서 의사소통, 단순함, 피드백, 용기의 네 가지 가치추구를 통해 소프트웨어의 생산성과 품질을 향상시키는 실질적인 소프트웨어개발방법론이다[1, 3, 5, 7]. XP에서 가장 중요한 실천사항(Practices) 중 하나인 테스트주도개발(Test Driven Development)은 실제코드를 작성하기 전에 테스트코드부터 작성하는 방식을 일컫는데, 테스트주도개발을 하지 않으면 XP를 한 것이 아니라고 말할 수 있을 정도로 XP에서 중요한 위치를 점하고 있다[2, 6].

하지만, 테스트주도개발에는 한 가지 잠재적인 문제가 있다. 테스트수행시간이 너무 오래 걸리거나 테스트케이스(Test Cases)가 아직 구현되지 않은 부분에 의존할 경우 테스트를 신속하고 독립적으로 실행할 수 없어 테스트주도개발이 불가능하다는 점이다[2]. 반복적인 테스트를 통해 소프트웨어결함을 재빨리 발견하고 수정하는 것이 테스트주도개발인데, 이와 같은 경우에서는 그 과정이 불가능하다. 예를 들어, 기능 A를 테스트하는 테스트케이스가 있고 기능 A는 데이터베이스로부터 특정 값을 받아 동작한다면, 데이터베이스를 구축하고 연결하기 전에는 기능 A를 테스트 할 수 없다.

이와 같은 테스트주도개발의 문제를 해결하기 위해 최근 모크객체의 중요성이 강조되고 있다. 모크객체는 문제가 되는 실제코드를 흉내내는 객체로서 수동으로 작성할 수 있을 만큼 간단한 구조를 가지는 것이 원칙이다[12, 15]. 그러나 실제로 모크객체를 수동으로 작성하는 것은 번거롭고 비효율적인 작업이기 때문에 자바(Java)에서는 MockObjects, EasyMock와 같은 모크객체생성기(Mock Objects Generators)를 사용한다[17, 18]. 문제는 모크객체가 기본적으로 객체지향을 전제로 하고 있고 아직 C언어용 모크객체생성기도 없기 때문에 C언어에서 모크객체를 사용하기가 까다롭다는 점이다. 이 때문에 테스트주도개발의 문제를 해결하는데 많은 노력이 들고, 이는 결국 C환경에 XP를 적용하는데 걸림돌이 되고 있다.

본 논문에서는 객체지향개념이 취약한 C언어에서 모크객체를 보다 쉽게 생성하기 위한 C언어용 모크객체생성기 CMock를 제시하고 그 효용성을 검증한다. 2장에서는 테스트주도개발과 모크객체에 대한 관련연구를 기술하였고, 3장에서는 기존의 모크객체생성기를 분석하여 요구사항을 도출하고 테스트코드로 구체화하여 CMock를 개발하는 과정을 기술한다. 4장에서는 개발한 CMock를 이용하여 테스트주도개발이 불가능한 경우를 해결한 후 테스트결과를 검토하고, 5장에서는 결론과 향후 연구를 정리한다.

Ⅱ. 관련연구

2.1 테스트주도개발

테스트주도개발은 K. Beck에 의해 소개된 것으로 본래 테스트우선프로그래밍(Test First Programming)이란 용어로 알려졌다. 이는 유닛테스트를 통해 테스트코드와 실제코드를 점진적으로 함께 발전시켜 소프트웨어를 완성하는 방식을 일컫는 것으로 XP에서 가장 기본적이고 중요한 실천사항이다[6, 13]. 테스트주도개발의 장점 중 하나로는 리팩토링(Refactoring)의 단점을 보완하는 역할을 들 수 있다. 리팩토링은 테스트주도개발보다 더 이전에 M. Fowler가 제안한 것으로 리팩토링을 통해 코드중복을 제거하고 보다 더 나은 설계를 점진적으로 해나갈 수 있다[8]. 하지만 리팩토링의 완료기준이 명확하지 않으면 무한반복에 빠질 위험이 있으며, 자칫 올바르지 못한 리팩토링 때문에 소프트웨어가 유연성을 잃을 수 있다. 테스트주도개발에서는 "중복이 없고 테스트를 통과하는 코드"를 목표로 삼아 리팩토링을 하고, 리팩토링의 결과를 지속적인 회귀테스트(Regression Tests)를 통해 신속하게 확인함으로써 리팩토링의 위험성을 감소시킨다[2, 4].

테스트주도개발은 이미 다양한 연구결과를 통해 그 효용성이 입증된 바 있다. M. M. Muller와 O. Hagner는 전통적인 프로그래밍과 테스트주도개발을 비교하는 실험을 수행하였는데, 19명의 대학원생을 구성하여 개발시간, 코드품질, 이해도 측면에서 전통적인 프로그래밍에 비해 테스트주도개발이 더 우월하다는 것을 증명했다[14]. 또한, B. George와 L. Williams는 테스트주도개발이 기존 폭포수적 실천사항(Waterfall-like Practices)에 비해 품질과 생산성 측면에서 더 나으며, BDUF(Big Design Up Front)의 단점을 근본적으로 보완할 수 있는 방법이라고 평가했다[11]. 다만, 아직 테스트주도라는 개념이 생소하고 기존 방식과는 다른 패러다임을 요구하기 때문에 현장에 적용하기에는 다소 진입장벽이 있으며, 이를 완화하기 위한 연구가 필요할 것으로 보인다.

2.2 모크객체

모크객체란 얼마나 자주 어떤 메서드가 어떤 인수로 호출되는지를 고려하여 기대값과 리턴값을 설정하고 검증하는 방법을 제공하는 클래스이다. 이는 기본적으로는 서버스텁(Server Stubs)과 유사하지만, 서버스텁은 전통적인 테스팅이론에 속하는 테스트이후접근법(Test After Approach)을 전제로 하고 있으며, 작성하기도 어렵고, 유지보수에 드는 비용과 실제 코드로 바꿀 때의 위험도 높다. 이에 비해, 모크객체는 테스트우선접근법(Test First Approach)에 따라 각 테스트를 철저히 분리시키고 모크객체가 서로 연결되는 것을 피함으로써 서버스텁이 가지고 있는 문제를 해결한다[2, 13].

이 모크객체 없이는 테스트주도개발이 불가능한 경우가 많은데, 그 이유는 실제객체가 다음과 같은 단점을 가지고 있기 때문이다[12].

- 실제객체의 행위는 결정되어 있지 않다.
- 실제객체는 설정하기 어렵다.
- 실제객체는 특정 상태를 유발시키기 어렵다.
- 실제객체는 느리다.
- 실제객체는 유저인터페이스와 관련이 있다.
- 실제객체는 어떻게 작동하는지 알아야 한다.
- 실제객체는 아직 존재하지 않을 수도 있다.

예를 들어, 실제객체의 행위가 결정되지 않은 경우에는 (그림 1)과 같은 문제가 발생한다.

```
import unittest, random
class DiceGame:
  def play(self, playerOne, playerTwo):
    if playerOne.rollDice() > playerTwo.rollDice():
      return playerOne
    return playerTwo

class DiceGameTest(unittest.TestCase):
  def testPlayerOneWin(self):
    diceGame = DiceGame()
    r = random
    dice = Dice(r)
    playerOne = Player(dice)
    playerTwo = Player(dice)
    self.assertEquals
    (playerOne, diceGame.play(playerOne, playerTwo))
...
```

그림 1. 테스트주도개발이 불가능한 코드
Fig. 1 Impossible Code in Test Driven Development

위 코드는 파이썬(Python) 언어로 작성된 코드로서, DiceGame클래스의 play(self, playerOne, playerTwo)는 playerOne과 playerTwo가 가지고 있는 rollDice()의 리턴값을 비교하여 승자를 판단한다. 이 기능을 테스트하는 테스트케이스 testPlayerOneWin(self)에서는 playerOne이 이기는 상황을 가정하고 있다. 이를 위해 랜덤값을 생성하는 파이썬모듈인 random을 이용하여 dice객체를 만든 후, 이를 Player 생성시에 인수로 넘겨준다. 만약, diceGame.play(playerOne, playerTwo)의 결과가 playerOne이라면, 테스트는 성공할 것이다. 그러나 랜덤값은 항상 변하는 것이므로 playerOne이 리턴될 수도 있고 아닐 수도 있다. playerTwo의 값이 더 크면 테스트는 실패할 것이다.

위와 같은 상황에서는 일관성 있는 테스트결과를 기대할 수 없다. 또한, 테스트케이스가 외부모듈에 의존하기 때문에 테스트케이스의 독립성 또한 지킬 수 없는 상황이다. 이 문제를 해결하기 위해 모크객체를 적용한 코드는 (그림 2)와 같다.

```
import unittest, mockobject
class DiceGameTest(unittest.TestCase):
  def testPlayerOneWin(self):
    r = MockRandom()
    r._setExpectedRandomValue(6)
    r._setExpectedRandomValue(5)
    r._verify()

class MockRandom(mockobject.MockObject):
  def __init__(self):
  def _setExpectedRandomValue(self, aValue):
    self._returnValue.addExpected(aValue)
  def randint(self, min, max):]
...
```

그림 2. 모크객체를 적용한 코드
Fig. 2 Code Applied on Mock Objects

import random과 r=random 부분을 import mockobject와 r=MockRandom()으로 대체했다. 모크객체인 MockRandom은 기대값을 설정하는 _setExpectedRandomValue(self, aValue)와 기대값을 리턴하는 randint(self, min, max)를 가지고 있다. 테스트코드에서는 MockRandom의 기대값을 6, 5로 설정한 다음, r._verify()에서 모크객체의 무결성을 테스트한다.

위 예를 통해, 모크객체가 테스트주도개발이 불가능한 상황을 어떻게 해결하는지를 알 수 있다. 즉, 실제객체가 가지는 한계 때문에 테스트를 진행하기 힘들 경우, 실제객체의 기대행위와 리턴값을 흉내내는 모크객체로 대체함으로써 쉽고 빠르게 특정 상태를 가정한 테스트를 진행할 수 있는 것이다.

그러나 모크객체가 아닌 실제객체를 이용하여 테스트를 해야 할 경우도 있다. 이럴 경우를 대비하여 모크객체를 사용한 테스트슈트와 실제객체를 사용한 테스트슈트로 나누는 것이 좋다. 일반적인 테스트주도개발을 할 때에는 모크객체를 사용하고 실제객체는 작업시간 이외에 자동으로 테스트하도록 하면 테스트주도개발에 무리를 주지 않으면서도 실제작동을 확인할 수 있는 것이다. 이처럼 모크객체는 테스트주도개발에 필수적인 기법으로서 XP에서 중요한 위치를 차지하고 있다.

III. 모크객체생성기 요구사항도출 및 개발

본 연구는 모크객체생성기의 기능요구사항을 도출하고 이를 테스트코드로 구체화한 다음, CMock를 개발하고 그 효용성을 검증하는 과정으로 이루어져 있다. 기능요구사항이란 소프트웨어가 가져야하는 기능에 대한 명세인데, XP에서는 이를 유저스토리(User Stories)로 표현한다. 유저스토리는 UP(Unified Process)의 유즈케이스(Use Cases)와 비슷한 개념이지만 덜 격식적이고 좀 더 간단하다는 차이가 있다.

XP에서 유저스토리는 테스트주도개발을 통해 테스트코드로 표현된다. 즉, 테스트코드가 해당 소프트웨어의 요구사항을 구체적으로 표현하는 실체이므로 기존 모크객체생성기의 테스트코드를 분석함으로써 모크객체생성기의 요구사항을 도출할 수 있다. 따라서 본 장에서는 CMock의 요구사항을 도출하기 위해 자바용 모크객체생성기인 MockObjects, EasyMock와 파이썬용 MockObjects의 테스트코드를 분석하였다.

3.1 MockObjects 분석

현재 가장 널리 쓰이고 있는 모크객체생성기 중 하나인 MockObjects는 풍부한 라이브러리를 바탕으로 세세한 모크객체컨트롤을 할 때 효과적이다. 자바환경을 지원하며, (그림 3)과 같은 구조를 가지고 있다.

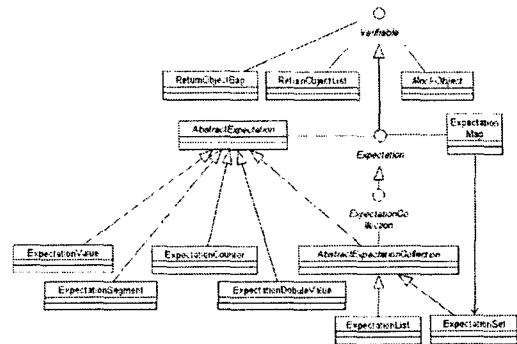

그림 3. MockObjects 클래스다이어그램
Fig. 3 A Class Diagram of Mock Objects

(그림 3)에서 Verifiable은 정확한 행위가 발생했는지 테스트종료 후에 확인하는 클래스의 인터페이스이며, MockObject는 모크객체에 사용되는 공통적인 메서드를 제공하는 추상클래스이다. Expectation은 Verifiable을 확장하는 인터페이스로서 이 클래스의 verify() 메서드는 기대값이 적합한지를 검사한다. Expectation Value는 기대값을 설정하는데 사용되며, Expectation Double Value는 double값에 대한 작동임을 제외하고는 Expectation Value와 목적이 비슷하다. Expectation Counter는 특정 상태가 몇 번이나 발생해야 하는지를 기술하는 기대값이고, Expectation List는 기대값과 실제값을 저장하고 있는 두 Array List의 인스턴스가 같은지를 확인할 수 있는 기대값이다. Expectation List는 기대값과 실제값을 저장하기 위해 두 개의 Hash Set 인스턴스를 사용하는 Abstract Expectation Collection의 구현이며, Expectation Map은 이름과 색인값으로 기대값을 관리하는데 사용된다.

Mock Objects는 〈표 1〉과 같이 총 15개의 테스트와 113개의 테스트케이스를 가지고 있다. 지원하는 기대값의 형태에 따라 테스트가 나뉘어 있으며, 해당 기대값에서 필요한 기능별로 테스트케이스가 구성되어 있다.

표 1. MockObjects 테스트목록
Table. 1 A Test List of MockObjects

테스트명	테스트 케이스 개수	대표적 테스트케이스
Test Assert Mo	13	test Assert Excludes
Test Expectation Collection	15	test Has Expectations
Test Expectation Counter	8	test Has No Expectations
Test Expectation Double Value	8	tes tPass On Error
Test Expectation List	1	test Sorted
Test Expectation Map	7	test Over write Entry
Test Expectation Segment	7	test FailsImmediately
Test Expectation Set	6	test Multi Unsorted
Test Expectation Value	17	test Int Pass
Test Map Entry	2	test Equals
Test Null	2	test Description
Test Return Object Bag	9	test Return Succeeds
Test Return Object List	4	test Too Many Returns
Test Return Value	7	tes tGet Value
Test Verifier	7	testInherit Verifiable Passes
총계	테스트 15개, 테스트케이스 113개	

위 테스트목록을 통해 MockObjects에서 어떤 기능을 제공하는지 알 수 있다. 예를 들어, TestExpectationValue에서 의도한 바와 같이 MockObjects는 자바에서 제공하는 자료형에 따라 기대값을 설정하는 기능이 있고, TestExpectationList처럼 다양한 형태의 기대값을 리스트로 묶는 기능을 가지고 있다. 즉, MockObjects에 따르면 모크객체생성기는 Collection, Counter, DoubleValue, List, Map, Segment, Set, Value형태의 기대값을 지원하고 이 기능을 일관적으로 묶는 내부기능으로 구성되어야 할 것이다.

3.2 EasyMock 분석

EasyMock는 프록시메커니즘(Proxy Mechanism)을 이용한 동적모크객체생성기능을 제공하기 때문에 〈표 2〉와 같이 MockObjects보다 테스트가 더 많다.

표 2. EasyMock 테스트목록
Table. 2 A Test List of EasyMock

테스트명	테스트 케이스 개수	대표적 테스트케이스
Arguments Matcher Test	5	test Setting The Same MatcherIs Ok
Arguments Test	2	test Equals
Array Matcher Test	12	testInt Array
Default Matcher Test	4	test Default Matcher
Equals Matcher Test	2	test Sorted
Legacy Behavior Tests	1	test Throw After Throwable
Matchable Arguments Comparison Test	4	test Equal Objects Equal To String
Matchable Arguments Test	2	test Equals
Method Call Test	2	test Equals
Object Methods Test	7	test Equals Before Activation
Record StateInvalid Default Return Value Test	5	test SetInvalid Default Float Return Value
Record StateInvalid Default Throwable Test	3	test Throw Null
Record StateInvalid Return Value Test	11	test SetInvalid Boolean Return Value
Record StateInvalid State Change Test	2	test Activate Without Return Value
Record StateInvalid Throwable Test	4	test Throw After Throwable
Record StateInvalid Usage Test	11	test Set Wrong Return ValueInt
총계	테스트 37개, 테스트케이스 294개	

위 테스트목록의 각 테스트케이스를 살펴보면, EasyMock 또한 MockObjects가 지원하는 거의 모든 형태의 기대값을 설정할 수 있음을 알 수 있다. 다만, 사용자가 모크객체를 쉽게 생성할 수 있도록 하기 위해 인터페이스에 대한 동적모크객체생성기능을 제공하기 때문에 내부로직은 다소 복잡하다.

3.3 CMock 요구사항도출

MockObjects와 EasyMock를 분석한 결과, MockObjects는 모크객체라이브러리의 집합이고 EasyMock는 여기에 동적모크객체생성기능을 추가한 것임을 알 수 있었다. CMock의 목적이 C환경에서 모크객체를 손쉽게 생성할 수 있도록 하는 것임을 감안하면 동적모크객체생성기능까지 포함해야 하지만, 이를 위해서는 객체지향의 인터페이스개념이 필요하기 때문에 C환경에는 적용하는 것이 무리가 있다고 판단

하는 바, EasyMock의 동적인 기능보다는 MockObjects의 기본적인 요구사항을 주로 수용할 것이다.

다만, 자바보다는 파이썬이 C언어와 유사한 형태로 프로그래밍이 가능하므로 파이썬용 MockObjects 또한 참조할 필요가 있다. 파이썬용 MockObjects를 이용하여 클래스를 제거한 형태로 모크객체를 사용한다면, C환경에서도 유사한 형태로 모크객체를 생성할 수 있을 것이다. 파이썬용 MockObjects는 자바용보다 다소 종류가 적은 Counter, List, Set, Map 형태의 기대값을 지원하는데, 이는 파이썬용 MockObjects의 기능적 취약점이라기보다는 파이썬언어 자체가 가지는 간결함에서 기인한 것으로 보인다.

이렇듯 자바용 MockObjects와 파이썬용 MockObjects에 기초하여 CMock의 테스트목록을 도출하면 〈표 3〉과 같다.

표 3. CMock 테스트목록
Table. 3 A Test List of CMock

테스트명	테스트 케이스명	테스트목적
Test Expectation Counter	test Expect Nothing	기대값을 생성한 뒤에 실제값을 비우고 기댓값 보유여부를 true로 설정하는지 테스트
	test Flush Actual	기댓값 보다 실제값이 먼저 설정되었을 경우 올바로 작동하는지 테스트
	test HasNo Expectations	기대값을 설정하지 않았을 때 올바로 작동하는지 테스트
	test Success	성공적으로 Expectation Counter를 사용한 경우를 테스트
Test Expectation List	test Sorted	리스트 내 요소가 올바른 순서대로 정렬되었는지 테스트
Test Expectation Set	test Multi Unsorted	기대값과 실제값을 정렬되지 않은 다수의 값으로 설정했을 때 올바로 작동하는지 테스트
	test Multi UnsortedSet	기대값과 실제값을 정렬되지 않은 다수의 모음값으로 설정했을 때 올바로 작동하는지 테스트
	test Unsorted	기대값과 실제값을 정렬되지 않은 값으로 설정했을 때 올바로 작동하는지 테스트
	test Unsorted Set	기대값과 실제값을 정렬되지 않은 모음값으로 설정했을 때 올바로 작동하는지 테스트
Test Expectation Map	test Expect One Entry	기대값과 실제값에 한 개의 값을 설정했을 때 올바로 작동하는지 테스트
	test Expect Two Entries	기대값에 실제값에 두 개 이상의 값을 설정했을 때 올바로 작동하는지 테스트
	test Overwrite Entry	key값이 중복된 경우에 "overwrite entry" 메시지가 뜨는지 테스트
총계	테스트 4개, 테스트케이스 12개	

위 테스트목록에 따르면, CMock는 Counter, List, Set, Map의 기대값을 지원하고, 총 12가지의 테스트케이스를 만족해야 한다. 위 테스트를 모두 통과한다면, CMock는 C 환경을 위한 모크객체생성기로서 기본적인 요구사항을 충족하고 있다고 말할 수 있을 것이다.

3.4 CMock 개발

CMock의 개발프로세스는 XP이다. 〈표 4〉와 같이 유닛테스트도구, 인수테스트도구, 버전관리도구, 빌드자동화도구, 협업도구가 필요하다.

표 4. CMock 개발에 필요한 도구
Table. 4 Tools for CMock Development

도구종류	해당 실천사항	도구명
유닛테스트도구	테스트주도개발, 디자인개선	CuTest
인수테스트도구	테스트주도개발, 계획 짜기게임, 완전한 팀	-
버전관리도구	공동코드소유권, 지속적인 통합, 작은 출시	Subversion
빌드 자동화도구	지속적인 통합, 작은 출시	-
협업도구	계획 짜기게임, 완전한 팀, 짝 프로그래밍	Media Wiki
기타		GCC, Dev-C++

위와 같이 유닛테스트도구와 버전관리도구, 협업도구는 마련되어 있다. 그러나 인수테스트도구와 빌드자동화도구는 없는데, 이는 자바에서 일반적으로 사용하는 FIT(Framework for Integrated Test), ANT에 준하는 C언어용 도구가 없었기 때문이다. 다만, 개발규모가 크지 않은 본 CMock 개발에서는 생략가능하다고 판단한다.

본 연구와 같이 요구사항에 따라 테스트케이스를 사전에 마련하고 이를 완성해나가는 방식은 XP에서는 BDUF라 하여 지양하고 있다[8, 10]. 테스트주도개발의 주목적은 요구사항의 명세화가 아니라 최적의 설계를 위한 실질적인 피드백이기 때문이다. 따라서 도출한 요구사항과 테스트만으로는 C환경에서 해결할 수 없는 문제가 발생한다면, 지속적인 테스트코드 추가로 이를 보완해 나가야 것이다.

IV. 모크객체생성기 효용성 검증

본 장에서는 개발한 CMock의 기능을 검증하기 위해 앞서 제시한 테스트주도개발이 불가능한 경우를 CMock로 해결하고 테스트결과를 통해 CMock의 효용성을 증명한다.

4.1 테스트코드작성

예제로 사용될 코드는 주사위게임이다. 랜덤값에 따라 승자를 판단하고 게임결과를 원격지의 저장소에 저장한다. 또한, 게임결과가 100이 넘어가면 가장 오래된 게임결과를 삭제한다. 여기서 원격지의 저장소란 데이터베이스일 수도 있지만, 본 예제에서는 네트워크로 연결된 컴퓨터에 있는 텍스트파일을 의미한다. 테스트코드는 〈표 5〉와 같이 9개의 테스트케이스로 이루어져 있다.

표 5. 테스트케이스목록
Table. 5 Test Case List

테스트케이스	설명
test Player One Win	플레이어1이 이기는 상황을 가정하여 테스트
test Player Two Win	플레이어2가 이기는 상황을 가정하여 테스트
test Roll	Dice 클래스의 roll() 메서드에 대한 테스트
test Record	결과가 원격지의 파일에 저장되는지 테스트
test Record Failure	원격지에 저장이 실패할 경우를 가정하여 테스트
test Read	원격지로부터 결과를 읽을 수 있는지 테스트
test Read Failure	원격지로부터의 읽기가 실패할 경우를 가정하여 테스트
test Remove	원격지로부터 결과 삭제가 가능한지 테스트
test Remove Failure	원격지의 결과삭제가 실패할 경우를 가정하여 테스트

test Player One Win, test Player Two Win, testRoll은 게임자체에 대한 테스트이고 test Record, test Read, test Remove는 원격지의 저장소와 관련된 테스트이며, 이들은 저장소와의 연결이 실패할 경우를 가정한 Failure 테스트를 포함한다. 위의 테스트를 테스트슈트로 묶는 코드는 (그림 4)와 같다.

```
CuSuite* DiceGameTestSuite(void)
{
  CuSuite* suite = CuSuiteNew();
  SUITE_ADD_TEST(suite, testPlayerOneWin);
  SUITE_ADD_TEST(suite, testPlayerTwoWin);
  SUITE_ADD_TEST(suite, testRoll);
  SUITE_ADD_TEST(suite, testRecord);
  SUITE_ADD_TEST(suite, testRecordFailure);
  SUITE_ADD_TEST(suite, testRead);
  SUITE_ADD_TEST(suite, testReadFailure);
  SUITE_ADD_TEST(suite, testRemove);
  SUITE_ADD_TEST(suite, testRemoveFailure);
       return suite;
}
...
```

그림 4. 테스트슈트코드
Fig. 4 Code Applied Mock Objects

위 코드는 CuTest에서 제공하는 방식에 따라 테스트슈트를 작성한 것이다. SUITE_ADD_TEST 부분에서 각 테스트케이스를 추가하고 suite를 반환한다. 이렇게 작성된 테스트슈트는 최종적으로 (그림 5)와 같이 AllTests.c로 통합된다.

```
#include <stdio.h>
#include "CuTest.h"
CuSuite* CuGetSuite();
CuSuite* CuStringGetSuite();
CuSuite* CMockTestSuite();
CuSuite* DiceGameTestSuite();
void RunAllTests(void)
{
  CuString *output = CuStringNew();
  CuSuite* suite = CuSuiteNew();
  CuSuiteAddSuite(suite, CuGetSuite());
  CuSuiteAddSuite(suite, CuStringGetSuite());
  CuSuiteAddSuite(suite, CMockTestSuite());
  CuSuiteAddSuite(suite, DiceGameTestSuite());
  CuSuiteRun(suite);
  CuSuiteSummary(suite, output);
  CuSuiteDetails(suite, output);
  printf("%s\n", output->buffer);
}
...
```

그림 5. AllTest.c 코드
Fig. 5 AllTest.c Code

테스트슈트는 모두 4가지이며, Cu Get Suite와 Cu String Get Suite는 Cu Test에 대한 테스트슈트이다. CMock Test Suite에서는 CMock가 모크객체생성기로서 요구사항을 만족하는지를 테스트하고, Dice Game Test Suite는 예제코드에 대한 테스트슈트이다. 이와 같이, 테스트케이스

를 계획하고 이를 실제코드로 구현한 후, 문제가 발생하는 코드에 모크객체를 적용하는 작업을 수행하였다.

4.2 모크객체생성

본 예제에서 필요한 모크객체는 크게 세 가지이다. 랜덤 값을 대체하는 Mock Random과 게임 결과 값을 기록할 Mock Repository, 네트워크역할을 대신할 Mock Network이다. 그 중에서 Mock Random과 관련된 테스트케이스인 test Roll()을 소개하면 (그림 6)과 같다.

```
void testRoll( CuTest* tc )
{
  MockRandom * pMockRandom;
  Dice * pDice;
  pDiceTest->CuTest = tc;
  pDiceTest->MockRandom =
  ( MockRandom * )malloc(sizeof( MockRandom ));
  MockRandomInit( pDiceTest->MockRandom );
  pMockRandom = pDiceTest->MockRandom;
  _setExpectedRandomValue(pMockRandom, 2);
  _setExpectedRandomValue(pMockRandom, 4);
  _setExpectedRandomValue(pMockRandom, 2);
  _setExpectedRandomValue(pMockRandom, 6);
  _setExpectedRandomValue(pMockRandom, 3);
  _setExpectedRa6ndomValue(pMockRandom, 1);
  pDiceTest->dice = ( Dice * )malloc(sizeof( Dice ));
  DiceInit(pDiceTest->dice, pMockRandom);
  pDice = pDiceTest->dice;
  CuAssertTrue(tc,2 != roll( pDice ));
  CuAssertTrue(tc,4 != roll( pDice ));
  CuAssertTrue(tc,2 != roll( pDice ));
  CuAssertTrue(tc,6 != roll( pDice ));
  CuAssertTrue(tc,3 != roll( pDice ));
  CuAssertTrue(tc,1 != roll( pDice ));
  _verify( pMockRandom->MockObject );
}
  ...
```

그림 6. 모크객체를 적용한 테스트코드
Fig. 6 Test Code Applied on Mock Objects

위 코드를 통해 알 수 있듯이 CMock로 모크객체를 생성하는 방법은 자바나 파이썬의 경우와 크게 다르지 않다. 기대값과 실제값을 설정한 후, 그 결과를 검증하는 과정이 동일하다. 다만, 객체지향을 지원하지 않는 C언어에서 변수와 함수를 논리적으로 묶기 위해 구조체와 포인터를 활용하고, 메모리관리를 직접 해야 하는 차이가 있다.

본 연구는 이와 같은 방법으로 테스트코드를 작성하고 모크객체를 생성한 후, 〈표 6〉과 같은 테스트환경에서 CMock의 효용성을 검증하는 테스트를 수행하였다.

표 6. 테스트환경
Table. 6 Test Environment

PC 종류	하드웨어 사양			
	CPU	메모리	하드	운영체제
PC 1	Pentium III 700MHz	256MB	40GB	Windows XP
PC 2	AMD Barton 2600+	1GB	240GB	Windows 2003
PC 3	Pentium IV 1.7GMHz	512MB	80GB	Windows XP
PC 4	Celeron 1.2MHz	128MB	40GB	Windows 98

테스트결과의 정확성을 위하여 테스트환경마다 20번의 테스트를 실행하였으며, 테스트환경도 다양화했다. 다만, 테스트코드가 높은 사양을 요구하지 않으므로 테스트환경의 차이가 CMock의 효용성 검증에 미치는 영향은 미미할 것으로 예상된다.

4.3 테스트결과

테스트결과는 〈표 7〉과 같이 모크객체를 사용하지 않은 경우, 모크객체를 수동으로 작성하는 경우, 모크객체생성기를 사용하는 경우로 나누었다. 해결여부(RP: Resolution Possibility)는 테스트주도개발이 불가능한 상황을 모크객체로 해결할 수 있는지를 가늠하는 척도로서 해결안됨(X), 부분해결(△), 해결가능(O)으로 표기하였다. 해결안됨이란 모크객체 없이는 테스트가 불가능하다는 것을 뜻하고, 부분해결은 특정 상태에 따라 테스트가 성공할 수도, 실패할 수도 있음을 의미한다.

개발시간(DP: Development Period)은 각 테스트케이스를 처음부터 새로 만들 때 소모되는 시간을 뜻하며 분으로 표기하였다. 코드라인(LOC: Line Of Code)은 해당 테스트케이스 하나만 있을 때의 코드라인수를 의미하고 표기법은 줄(Line)이다. 테스트수행시간(TRT: Test Running Time)은 초로 표기하였는데, CuTest에는 테스트수행시간을 출력하는 기능이 없었기 때문에 같은 기능을 하는 파이썬코드에서 테스트수행시간을 측정하였다. C가 파이썬보다 실행속도가 빠른 만큼, 실제 테스트수행시간은 아래 결과보다 다소 빠를 것으로 예상된다. 또한, 각 테스트환경마다 테스트수행시간이 크게 차이가 나지 않았기 때문에 평균값으로 산정하였다.

표 7. 테스트결과
Table. 7 Test Results

테스트 케이스	모크객체를 사용하지 않은 경우				수동으로 작성하는 경우				모크객체생성기를 사용하는 경우			
	RP	DP	LOC	TRT	RP	DP	LOC	TRT	RP	DP	LOC	TRT
test Player One Win	X	32m	49L	0.031s	O	16m	95L	0.025s	O	17m	52L	0.016s
test Player Two Win	X	10m	47L	0.029s	O	9m	91L	0.026s	O	8m	53L	0.015s
test Roll	X	42m	40L	0.032s	O	35m	92L	0.052s	O	28m	51L	0.034s
test Record	△	95m	123L	0.621s	O	56m	152L	0.043s	O	48m	138L	0.033s
test Record Failure	△	24m	154L	0.634s	O	18m	169L	0.059s	O	17m	173L	0.042s
test Read	△	72m	131L	0.532s	O	42m	171L	0.045s	O	46m	156L	0.052s
test Read Failure	△	27m	160L	0.594s	O	21m	201L	0.051s	O	15m	182L	0.060s
test Remove	△	88m	128L	0.587s	O	62m	165L	0.048s	O	52m	142L	0.032s
test Remove Failure	△	18m	141L	0.601s	O	21m	191L	0.060s	O	16m	163L	0.045s
평균	△	45m	104L	0.406s	O	31m	147L	0.045s	O	29m	123L	0.037s

CMock의 효용성을 보다 명확하게 나타내기 위해 테스트결과를 비교하면 <표 8>과 같다. 모크객체를 수동으로 작성하는 경우와 모크객체생성기를 사용하는 경우를 모크객체를 사용하지 않은 경우와 비교하였으며 수치가 높을수록 효과가 큰 것을 의미한다.

표 8. 테스트결과비교
Table. 8 Test Results Comparison

테스트케이스	수동으로 작성하는 경우				모크객체생성기를 사용하는 경우			
	RP	DP	LOC	TRT	RP	DP	LOC	TRT
test Player One Win	100%	50%	-93.9%	19.4%	100%	46.9%	-6.1%	48.4%
test Player Two Win	100%	10%	-93.6%	10.3%	100%	20%	-12.8%	48.3%
test Roll	100%	16.7%	-130%	-62.5%	100%	33.3%	-27.5%	-6.3%
test Record	100%	41.1%	-23.6%	93.1%	100%	49.5%	-12.2%	94.7%
test Record Failure	100%	25%	-9.7%	90.7%	100%	29.2%	-12.3%	93.4%
test Read	100%	41.7%	-30.5%	91.5%	100%	36.1%	-19.1%	90.2%
test Read Failure	100%	22.2%	-25.6%	91.4%	100%	44.4%	-13.6%	89.9%
test Remove	100%	29.5%	-28.9%	91.8%	100%	40.9%	-10.9%	94.5%
test Remove Failure	100%	-16.7%	-35.5%	90.0%	100%	11.1%	-15.6%	92.5%
평균	100%	31.1%	-41.3%	88.9%	100%	35.6%	-18.3%	90.9%

이러한 테스트결과를 통해 알 수 있는 것은 크게 세 가지다. 첫째는 테스트주도개발이 불가능한 경우에 모크객체를 사용하면 모두 해결할 수 있다는 점이다. 물론, 모크객체를 사용하지 않아도 해결가능한 경우도 있었으나, 테스트결과가 일관성이 없었다. 문제가 되는 경우를 모두 해결할 수 있었기 때문에 효과는 100%로 볼 수 있다.

둘째는 모크객체를 사용하면 코드라인은 다소 늘어나지

만, 테스트수행시간은 크게 줄어든다는 점이다. 특히, 여기서 말하는 테스트수행시간은 평균값이기 때문에 모크객체를 사용하지 않은 경우에는 그 편차가 심했다. 코드라인이 평균적으로 -41.3%, -18.3%만큼 부정적인 효과를 보이는데 반해, 테스트수행시간은 88.9%, 90.9%만큼의 긍정적인 효과를 얻었다.

셋째는 모크객체생성기를 사용하면 모크객체를 수동으로 작성하는 것보다 개발시간과 코드라인이 줄어든다는 점이다. 모크객체를 수동으로 작성할 경우 개발시간은 31.1%, 모크객체생성기를 사용하면 개발시간은 35.6%, 즉 4.5%만큼의 긍정적인 효과가 있었으며, 코드라인 또한 -41.3%에서 -18.3%까지 부정적인 효과가 줄어드는 것을 알 수 있다. 이는 모크객체생성기가 필요한 이유를 설명해주는 부분이다.

V. 결론 및 향후연구

본 논문에서는 C언어용 모크객체생성기인 CMock를 개발하고 이의 효용성을 검증하였다. 검증결과, 특정 경우를 설정하고 테스트결과의 일관성을 확보하기 위해 필요한 모크객체를 CMock를 통해 손쉽게 생성함으로써 테스트주도개발에서 문제가 되는 상황을 신속히 해결할 수 있었다. 이로써 얻을 수 있는 CMock의 이득을 크게 두 가지로 정리할 수 있다.

첫째, 모크객체를 사용하지 않을 때 발생하는 테스트주도개발의 문제점을 C환경에서 해결할 수 있게 되었다. 실제객체는 특정 상태를 설정하기 어렵고 아직 존재하지 않을 수도 있으며 또한 느리므로 실제객체만으로는 테스트주도개발이 불가능한 경우를 해결할 수 없다. CMock는 C환경에서 실제객체를 모크객체로 대체하는 일관성 있는 방법을 제공함으로써 테스트주도개발의 문제점을 해결가능하게 한다.

둘째, 모크객체를 수동으로 작성할 때보다 개발시간과 코드라인을 줄일 수 있게 되었다. 비록 가능한 한 단순한 구조를 가지는 것이 모크객체의 원칙이긴 하지만, 모크객체를 수동으로 작성하는 것은 시간을 많이 소모하는 작업이므로 생산성 측면에서 부정적이다. 또한, 모크객체의 기본적인 기능을 각 모크객체마다 작성해야 하므로 코드라인마저 늘어나게 된다. CMock는 C환경에 필요한 모크객체라이브러리를 제공함으로써 모크객체사용에 소모되는 불필요한 작업을 감소시킨다. 따라서 CMock는 C환경에 XP를 적용하고자 할 때, 필수적이고 유용한 도구가 될 것이다.

그러나 CMock는 자바용 모크객체생성기에 비해 기능적 측면에서 부족한 점이 있다. 모크객체라이브러리가 풍부하지 않고 동적생성기능 또한 미흡하다. 이는 지속적인 유지보수로 해결해 나갈 예정이다.

모크객체를 C언어에서 사용하기보다는 파이썬과 같이 쉽게 모크객체를 사용할 수 있는 언어를 개발프로젝트에서 병용함으로써 기능을 우선 완성하고 최적의 성능이 필요한 부분만 C언어로 변환하는 것이 보다 효과적인 방법이다. 하지만 파이썬을 이용하지 못하는 임베디드소프트웨어개발과 같은 특정한 분야에서는 본 연구에서 제시한 CMock가 유용할 것이다. 향후에는 Field Experiment를 통해서 CMock를 검증할 필요가 있으며, C언어용 인수테스트도구에 대한 연구도 수행해야 할 것으로 보인다.

참고 문헌

[1] A. Cockburn, Agile Software Development, Addison Wesley, 2001.
[2] D. Astels, Test Driven Development: A Practical Guide, Prentice Hall PTR, 2003.
[3] J. Highsmith, Adaptive Software Development, Dorset House, 1999.
[4] J. Link, Unit Testing in Java, Morgan Kaufmann, 2003.
[5] K. Beck, Extreme Programming Explained: Embrace Change, Addison Wesley, 2000.
[6] K. Beck, Test Driven Development: By Example, Addison Wesley, 2003.
[7] K. Beck, M. Fowler, and J. Kohnke, Planning Extreme Programming, Addison Wesley, 2000.
[8] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison Wesley, 1999.
[9] R. Jeffries, A. Anderson, and C. Hendrickson, Extreme Programming Installed, Addison Wesley, 2001.
[10] S. Giancarlo, and M. Michele, Extreme Programming

Examined, Addison Wesley, 2001.

[11] B. George, and L. Williams, "An Initial Investigation of Test Driven Development in Industry," ACM Symposium on Applied Computing, pp.1135-1139, Mar., 2003.

[12] D. Thomas, and A. Hunt, "Mock Objects," IEEE Software, Vol. 19, No. 3, pp. 22-24, May/June, 2002.

[13] K. Beck, "Aim, fire (test-first coding)," IEEE Software, Vol. 18, No. 5, pp. 87-89, Sept./Oct., 2001.

[14] M. M. Muller, and O. Hagner, "Experiment about Test-first Programming," IEE Proceedings Software, Vol. 149, No. 5, pp. 131-136, Oct., 2002.

[15] T. Mackinnon, S. Freeman, and P. Craig, "Endo-Testing: Unit Testing with Mock Objects," Extreme Programming and Flexible Processes in Software Engineering, June, 2000.

[16] L. Williams, and R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the case for pair programming," IEEE Software, Vol. 17, No. 4, pp. 19-25, July/Aug., 2000.

[17] EasyMock, http://www.easymock.org/Documentation.html.

[18] MockObjects, http://www.mockobjects.com/MocksObjectsPaper.html.

[19] 임동주, 최호진, "객체지향 프로그램의 클래스 테스팅," 한국컴퓨터정보학회논문지, 5권, 2호, 2000.

[20] 임근, 권영만, "객체모델에 대한 형식명세로의 변환방법," 한국컴퓨터정보학회논문지, 8권, 4호, 2003.

저 자 소 개

정 영 목

2003년 학점은행제 전자계산학 (이학사)

2003년~현재 숭실대학교 대학원 컴퓨터공학과 석사과정

〈관심분야〉 Extreme Programming XInternet/RIA, Python

박 제 원

2004년 상지대학교 컴퓨터정보공학부 (공학사)

2004년~현재 숭실대학교 대학원 컴퓨터공학과 석사과정

〈관심분야〉 소프트웨어프로세스, 테스팅기술, 임베디드시스템

이 남 용

1979년 숭실대학교 컴퓨터학부 (이학사)

1983년 고려대학교 경영대학원 경영정보학 (경영학석사)

1993년 미시시피주립대학교 경영정보학 (경영학박사)

1999년~현재 숭실대학교 컴퓨터학부 교수

〈관심분야〉 객체기술(UML, RUP), 컴포넌트기술(OOAD, CBD), 테스팅기술