객체지향 시스템 개발에서의 유스 케이스 활용성에 관한 연구

정 승 렬* · 임 좌 상**

Investigating the Practice of the Use Case Modeling in the Object-Oriented Systems Development

Seung Ryul Jeong* · Joa Sang Lim**

Abstract

Object-oriented paradigm has required the fundamental changes in the developers' practice and orientation. However, the relevant literature has not still provided the solid practical methodology but created some myths regarding Use Case modeling. In this study, we attempt to "demythologize" some key aspects of Use Case Modeling by describing what we have observed in our research and practice. In order to identify the developers' perceptions, the survey methodology was utilized. By discussing the differences between what must be done and what has been done, this study provides rich insights into the better understanding of the Use Case modeling and the successful development of object-oriented systems.

Keywords : Use Case, Use Case Modeling, Object-Oriented Systems Development

논문접수일 : 2004년 4월 28일 논문게재확정일 : 2004년 10월 6일

* 국민대학교 비즈니스IT 전문대학원 교수
** 상명대학교 소프트웨어대학 미디어학부 교수

1. 서 론

객체지향(Object Orientation)은 1960년대 이후 처음 소개되었는데 재사용과 유지보수 측면에서 이른바 소프트웨어 위기를 극복하기 위한 대안으로 고려되면서 1980년대에는 분석설계 방법으로[Jacobson et al., 1992 Rumbaugh et al., 1991], 그리고 1990년대에는 Java 언어를 중심으로 전사적인 규모의 개발 방법으로 그 활용범위가 확대되고 있다. 이와 같이 객체지향이 급속하게 확산된 배경에는 UML과 같은 표준이 큰 공헌을 하고 있는데, 유스 케이스(Use Case)는 UML의 일부로서 대부분의 객체지향 방법론에서 활용되고 있다. 하지만 유스 케이스를 실무에 적용하는 문제는 그리 용이하지 않다는 것이 여러 문헌에서 지적되고 있다. 특히 유스 케이스 명세의 내용과 관련하여 아직도 이견이 많고, 그 유용성도 투입공수에 비해 적은 것으로 나타나고 있다. 결국, 이러한 결과는 객체지향 시스템 개발 방법론에서 활용되는 유스 케이스에 대해 보다 체계적인 접근과 연구가 필요하다는 것을 반증하고 있다. 본 논문은 객체지향 프로젝트에서 유스 케이스를 적용하면서 실무자들이 겪는 어려움이 많은데 반해 실제 활용방식에 관한 체계적인 연구가 적다는 점에 착안하여, 유스 케이스 모델링의 활용실태를 조사하고, 문제점과 개선방향을 제시하는 것을 목적으로 한다. 따라서 한국의 객체지향 프로젝트 실무자를 대상으로 먼저, 유스 케이스의 인식과 관련한 난이도와 크기를 조사해 보고, 두 번째로 유스 케이스 명세의 작성과 관련한 난이도, 유용성, 작성 내용에 관한 실태를 살펴본다. 다음 장인 2장에서는 유스 케이스 모델링에 관해 보다 상세히 살펴보고, 그 다음 장에서는 연구방법에 대해 설명한다. 4장에서는 설문을 통해 얻어진 결과를 분석하고 마지막으로 논문에서 얻어진 결과를 토론한다.

2. 이론적 배경

2.1 유스 케이스

유스 케이스는 정보시스템 사용자와 정보시스템 간의 상호작용에서 발생할 수 있는 모든 시나리오를 문서화한 것으로, Objectory 방법론[Jacobson et al., 1992]에서 처음 제안되었다. 그 후 유스 케이스는 객체지향 시스템을 만드는데 출발점으로 인식되어 현재 대부분의 객체지향 개발에서 적용되고 있으며, UML의 표준으로 채택되었다. 유스 케이스를 인식하는 방법으로 RUP(Rational Unified Process)에서는 먼저 정보시스템의 사용자인 액터(actor)를 인식하고, 그 후 액터의 목적 및 역할을 파악하여, 그 목적에 부합하는 유스 케이스를 인식한다. 그러나 액터를 인식하고 기능을 인식한 후 관련 시나리오를 묶어서 유스 케이스를 인식하는 방법도 채택할 수 있다. 유스 케이스는 이벤트로 이루어져 있고, 여기서 클래스가 인식되며, 유스 케이스 내의 시나리오는 향후 순차 다이어그램으로 연결된다. 또한 구현할 때는 컨트롤러 패턴(controller pattern)에 따라 사용자의 이벤트를 받아서 시스템에 연결하는 기능을 수행한다. 유스 케이스는 그 형식에 따라서 분석, 설계로 구분할 수도 있다.

유스 케이스는 사용자가 시스템을 사용하는 사례를 묶어 놓은 것이기 때문에, 사용자의 관점에서 시스템을 구현할 수 있다는 장점이 있다. 이러한 사용자 관점에서 시스템을 설계하겠다는 철학으로 인해 유스 케이스는 기능의 절차를 포함하는 것이 일반적이다. 시작과 끝이 있다는 점에서 프로세스 모델링에서 말하는 기본업무 프로세스 (Elementary Business Process)

와 비교 되기도 한다[Nurcan et al., 1998]. 즉, 프로세스는 사용자의 가치창출을 설계의 기준으로 삼고 있다는 점에서 유스 케이스와 같은 목적을 갖고 있다. 유스 케이스를 마치게 되면 사용자는 원하는 목적을 달성하고, 이로 인해 만족하게 된다. 물론 유스 케이스는 여러 시나리오를 담게 되며 따라서 시나리오와 혼돈될 수도 있다. 하지만 하나의 유스 케이스는 보통 1개 이상의 시나리오를 담고 있고 각 시나리오는 테스트 사례를 작성하는데 기준이 되기도 한다. 즉, 유스 케이스는 동일한 사건에 대해 사용자가 시스템을 사용하면서 발생할 수 있는 여러 시나리오를 갖는다. 예를 들어, 로그인 유스 케이스의 경우, 로그인 성공, 로그인 실패, 로그인 ID 중복 등 많은 시나리오가 존재한다. 기능적 방법론에서는 요구사항이 정의된 후, 구현 단계에 가면 그 추적이 어려워지는 경향이 있지만, 유스 케이스는 오히려 프로젝트가 진행됨에 따라 진화하면서 발주자에게는 인수에 필요한 추적성을, 그리고 개발자에게는 개발문서의 시작 문서로서 활용성을 인정받고 있다.

2.2 유스 케이스 모델링

유스 케이스 모델링은 객체지향 시스템 개발에 있어 가장 중추적인 역할을 한다. 유스 케이스 모형은 시스템 개발에 있어 근본이 되는 것이며, 클래스를 인식하는 원천이면서 시스템 분석 및 설계, 그리고 시험단계의 모형을 전체적으로 제어한다. 결국, 유스 케이스는 사용자의 요구가 다음 단계의 모형에 어떻게 반영되고 있는지를 추적할 수 있도록 하는 통제도구 역할을 하는 것이다. 이와 같이 중요한 유스 케이스에 대해 개발자들이 정확하고 올바른 유스 케이스 모델링을 수행하지 못한다면 객체 기반의 시스템 개발은 매우 우려할 만한 결과를 가져오게 될 것이다.

실제로 객체 프로젝트에서 유스 케이스를 적용하다 보면, 폭포수 방식에 익숙한 개발자에게는 유스 케이스 모형이 이해하기 어렵고, 개발에 활용하기에는 너무 단순한 문서로 이해되고 경시되는 풍조가 있다. 특히 서술식으로 문서를 작성하는 방식에 익숙하지 않은 개발자들이 유스 케이스 명세와 같은 문서를 만들다 보면, 클래스의 기반으로 활용하기도 어려워 지는 단점이 나타나기도 한다.

기존 문헌들에서도 위에서 언급한 것처럼 유스 케이스 모델링이 쉽지 않음을 알 수 있다. Iivari and Maansaari[1998]는 객체지향을 적용하는데 있어서의 기술적 어려움을 지적하였으며 Sheetz & Tegarden[2001]은 개발자들이 겪는 인지적인 어려움을 제시하였다. 특히 Fowler [2003]는 많은 경우 개발자들의 객체지향적 설계 결과가 기존의 전통적 방식과 차별화 되지 않음을 비판하기도 하였다.

본 논문에서는 객체지향과 대비되는 비객체지향적 접근으로 기능적 방법이라는 용어를 사용하였다. 또한 소프트웨어 개발주기가 단계별로 반복되면서 점진적으로 시스템이 구현되는 방식을 반복적 방법이라고 지칭하였고, 반면에 개발주기가 반복되지 않는 전통적 방법을 폭포수 모형이라고 하였다. 반복적 방법에서 각 단계는 착수(Inception), 상세(Elaboration), 구현(Construction), 전환(Transition)이라고 명명되었다. 특히, 클래스에 캡슐화되는 요소는 많은 용어가 혼용되고 있지만, 본 논문에서는 속성(Attribute)과 방법(Method)이라는 용어를 사용하였다. 유스 케이스는 Use Case를, 유스 케이스 명세는 Use Case Specification을 지칭하는 용어로 사용되었다.

3. 데이터 수집 및 응답자 특성

본 논문에서는 개발자들이 유스 케이스와 관

련하여 어떻게 생각하고 있는지 그리고 어떻게 수행하고 있는지를 살펴보기 위해 설문서를 이용한 현장연구 방법을 택하였다.

또한 유스 케이스에 관한 인식 및 태도를 실무적인 관점에서 살펴보기 위해 응답자를 객체지향 프로젝트를 수행한 경험자로 한정하였다. 즉, 응답자는 실무 프로젝트에서 유스 케이스를 작성하였거나, 유스 케이스를 통하여 클래스 및 순차 다이어그램을 설계하였거나, 유스 케이스를 참조하여 소스코딩을 하는 등 유스 케이스를 실무에서 충분히 경험한 인력들로 구성하였다. 설문은 순차적, 명목적 유형으로 구성하였는데, 순차적 유형은 리커트 5점 척도의 설문으로 1번은 매우 긍정, 3번은 보통, 5번은 매우 부정을 의미하였다. 명목적 유형은 응답자가 해당 항목에 의견을 표시하는 것으로 하였다. 총 50명의 개발자에게 설문을 보냈으며, 27명이 참여하여 54%의 설문 응답율을 보였다. 응답자들의 특성을 살펴보면 우선 IT 프로젝트의 경험도 측면에서 3분의 2 이상이 2년 이하로 나타났다(<표 1> 참조). 특히 과반수가 1년에서 2년 사이의 경험자들로 나타났으며 5년 이상의 숙련 개발자는 약 11% 정도로 나타났다. 일반적인 IT 개발자와 비교해서 응답자들의 개발 경력이 비교적 짧은 것으로 나타난 이유는 아마도 젊은 층이 많은 객체 프로젝트를 경험한 개발자 그룹에서 표본을 선정하였기 때문으로 풀이된다.

〈표 1〉 응답자의 프로젝트 경력

기 간	빈 도	비 율
-1년	6	22.2
-2년	13	48.1
-3년	2	7.4
-4년	1	3.7
-5년	2	7.4
5년 초과	3	11.1
총 응답자 수	27	100.0

한편 응답자들의 개발 언어 경험정도(<표 2> 참조)를 살펴보면 예상한대로 대부분인 약 93%가 Java를 경험한 것으로 나타났다. 하지만 COBOL이나 PowerBuilder, Delphi 경험자 들이 적다는 사실은 객체 개발자들이 전통적 시스템이나 클라이언트 서버 시스템 개발자 출신이 아니며 오히려 웹 어플리케이션이나 객체 시스템 분야에서 새롭게 개발 경력을 시작한 사람들이 많음을 암시하고 있다.

〈표 2〉 응답자의 개발경험 언어

언 어	빈 도	비 율
C	5	18.5
Java	25	92.6
Power Builder	3	11.1
Delphi	2	7.4
COBOL	3	11.1
C++	1	3.7
Visual Basic	6	22.2
기타	8	29.6
총 응답자 수	27	복수 응답 가능

객체 프로젝트의 경험 정도를 살펴보면 아래 <표 3>에서 보여지듯이 대부분의 응답자가 1년 반 미만의 경험을 보유하고 있음을 알 수 있다. 이는 객체 시스템에 대한 관심이 고조되기 시작한 시점이 그리 오래지 않음을 감안해 볼 때 당연한 현상으로 파악되며 따라서 실제 현장에서 오랜 기간 객체 프로젝트를 경험한 고급 개발자들이 그리 많지 않음을 알 수 있다.

한편 이들 중 컴포넌트 프로젝트의 경험 정도는 어떤지 알아보았다. 여기서 컴포넌트 프로젝트란 EJB, COM, CORBA 중 하나의 기술을 적용한 경우를 의미한다. <표 4>에서 보듯이 약 70% 정도는 컴포넌트 프로젝트를 수행한 경험이 있는 것으로 나타났다. 이들 컴포넌트 시스템 유 경험자들의 경험 기간(<표 5> 참조)을

살펴보면 대체로 객체 프로젝트 경험기간의 패턴과 유사한 것으로 나타났다. 다만 컴포넌트 프로젝트의 경험이 객체 프로젝트에서 보다 조금 더 짧은 것을 알 수 있다.

〈표 3〉 응답자의 객체지향 프로젝트 경험

기 간	빈 도	비 율
-6개월	6	22.2
-12개월	10	37.0
-18개월	8	29.6
-24개월	2	7.4
24 개월 초과	1	3.7
총 응답자 수	27	100.0

〈표 4〉 응답자의 컴포넌트 프로젝트 경험

	빈 도	비 율
있 음	19	70.4
없 음	8	29.6

〈표 5〉 응답자의 프로젝트 수행 경력

기 간	빈 도	비 율
-6개월	9	47.4
-12개월	5	26.3
-18개월	4	21.1
-24개월	0	0.0
24 개월 초과	1	5.3
총 응답자 수	19	100.0

4. 분석 결과

4.1 객체지향 프로젝트에 대한 인식

개발자들의 객체지향 프로젝트의 품질에 대한 인식을 살펴보기 위해 (1) 유스 케이스 모델링의 적정성, (2) 설계의 적정성, (3) 전반적인 프로젝트의 성공도에 관해 물어보았다(표6 참조). 유스 케이스 모델링의 적정성 및 프로젝트 성공도에 대한 평가는 3.07로 중립적인 의견(3점)에 근접하였으나 설계가 객체지향적으로 적정하였는가 하는 질문에 대해서는 평균에 조금 미치지 못하는 부정적인 인식(3.30)을 보였다. 즉, 설계가 객체지향적으로 적절히 수행되었다고 동의하기 어렵다는 의견을 보였는데, 그 이유로 우선 객체지향의 난이성을 꼽을 수 있다. Sheetz & Tegarden[2001]에 따르면 객체지향이 일상생활에서 접하는 명사를 클래스로 만들기 때문에 자연스럽다는 주장과는 달리 인지적으로 많은 어려움을 수반한다고 한다. 이러한 인지적인 어려움이 실무에서 객체지향을 적용하는데 장애요인이 되는 것으로 보이는데, 1990년대 후반 핀란드에서 실시된 설문조사에 따르면 객체지향이 과거에 비해 실무에 적용되는 사례가 증가하고는 있지만, 아직 기대에 못 미치는 이유로 기술적 어려움을 꼽고 있다[Iivari and Maansaari, 1998]. 두 번째는 '혼합적인 기술' 측면에서 그 원인을 살펴볼 수 있는데, 객체지향을 적용하면서 데이터베이스는 비객체지향적인 관계형으로 구현한다는 점이다. 이로 인해 도메인에서 클래스를 인식하기보다는 테이블의 엔티티를 클래스로 인식하는 방식을 취하고 있고 따라서 이러한 방식을 객체지향적 설계라고 보기에는 기존 방식과의 차이가 크지 않아 어렵다는 점이다[Fowler, 2003]. 마지막으로 프로젝트의 일정을 감안하여 위험을 감수하기 보다는 검증되고 안전한 기존 방식으로 회귀하는 태도를 꼽을 수 있다. 즉, 객체지향에서 강조하고 있는 재사용성을 높이기 위해 상속, 클래스의 책임분할과 같은 기술적 방식보다는 과거에 그들이 해왔던 방식(예 : Stored Procedure)으로 위험을 감수하려 하지 않는 태도가 객체지향적인 설계를 저해할 수 있다

〈표 6〉 객체지향 프로젝트에 대한 인식

	평 균	표준 편차
선택한 객체지향 프로젝트에서 유스 케이스 모델링은 적정하였다.	3.07	0.73
선택한 객체지향 프로젝트는 객체지향적으로 설계되었다.	3.30	0.78
선택한 객체지향 프로젝트는 매우 성공적이었다.	3.07	0.62

4.2 유스 케이스 모델에 관한 인식

유스 케이스에 대한 인식을 살펴보기 위해 표 7의 항목을 물었는데, 개발자들은 유스 케이스가 기능분해도의 상위기능 또는 하위기능과 차이가 있다는 주장에 분명하게 긍정하는 태도를 유보했으며 (2.81), 유스 케이스를 도출하는 것이 기능을 도출하는 것 보다 어렵다고 응답하였다(2.11) 이러한 태도는 개발자들이 (1) 상위기능에서 하위기능으로 구체화하는 종전의 DFD (Data Flow Diagram)적인 사고에 익숙해져 있고, (2) 유스 케이스가 객체지향적인 시스템의 구현에 미치는 역할을 충분히 이해하지 못한 것으로 유추할 수 있다. 문헌에서는 유스 케이스가 기능을 계층화하는 수단으로는 적합하지 않으며, 사용자가 목표로 하는 가치를 만족시켜주기 위한 시나리오로서 시작과 끝이 있도록 인식되어야 한다고 한다[Cockburn, 2001 ; Jacobson et al., 1995]. 물론 기능과 유스 케이스에는 이론적으로 분명한 차이가 있지만, 이러한 정의는 과거에 개발자들이 익숙한 기능과 비교하여 상당한 애매모호성을 내포하고 있어 유스 케이스를 인식하는데 상당한 인지적 노력이 요구되고 있다. 이러한 어려움을 최소화하기 위해 저자들은 실무에서 보다 이해하기 쉬운 기능을 우선 인식하고, 목적이 같은 경우 기능을 시나리오로 묶어서 유스 케이스로 인식하는 이른바 상향식 접근을 적용하여 보았는데 보다 쉽게 적용될 수 있음을 경험한 바 있다.

〈표 7〉 유스 케이스에 대한 인식

	평 균	표준 편차
유스 케이스는 기능분해도의 상위기능 또는 하위기능과 차이가 없다.	2.81	1.00
유스 케이스를 도출하는 것이 기능을 도출하는 것보다 더 어렵다.	2.11	1.01
구현단계에서 어떤 유스 케이스가 어떤 클래스를 도출해서 구현했는지 그 추적성이 확보되어야 한다.	2.26	0.71
유스 케이스의 갯수는 많으면 많을수록 좋다.	3.30	1.03

한편 유스케이스가 사용자의 요구사항이 어떤 클래스에 구현되었는지 추적성을 제공해야 한다는 설문에는 개발자들이 동의하는 태도를 보였다(2.26)[Lindvall, 1994]. 유스 케이스는 도메인에서 사용되는 개념과 기능을 포함하고 있어서, 실무에서는 명사(구)를 인식하여 이른바 언어학적인 방식으로[Larman, 2002] 클래스를 인식하여 추적성을 확보하며, 전문적인 도구를 사용하기도 한다(예 : Requisite Pro)[Leffingwell & Widrig, 2000]. 반면, 개발자들은 유스 케이스가 많을수록 좋은가라는 질문에는 약간 부정적인 태도(3.30)를 보였다. 표 8에서 보듯이, 전사적인 프로젝트에 대해 적정한 유스 케이스의 개수는 30%가 20개 이하를, 약 15%는 30~60개가 적정하다고 응답하였다. 한편 100~200개가 적정하다고 생각하는 개발자도 약 30% 정도나 되어 유스 케이스의 크기와 개수에 대해 아직 공감대가 형성되지 못한 것으로 보인다. 이는 보통 객체지향 개발방법론에서도 아직 합의가 안되어 있는 것으로 나타난다[Regnell et al., 1996].

〈표 8〉 전사적 프로젝트에서의 적당한 유스 케이스의 수

항 목	빈 도	비 율
5	3	11.1
10	2	7.4
20	3	11.1
30	1	3.7
50	2	7.4
60	1	3.7
100	4	14.8
150	3	11.1
200	1	3.7
무응답	7	25.9
총 응답자 수	27	100.0

유스 케이스의 크기에 대한 인식을 구체적으로 알아보기 위해 각 유스 케이스에서 도출될 화면의 수에 대해 의견을 물었다. 표 9에 나타나듯이 약 82%의 개발자들은 하나의 유스 케이스에서 2~3개의 화면이 도출되는 것이 적정하다고 느끼고 있었다. 저자들이 참여했던 프로젝트에서는 데이터의 입력, 수정, 삭제, 조회와 관련하여 보통 2~3개의 화면이 사용되었다는 점을 감안하면 유스 케이스의 적정한 크기를 계량화하는데 도움이 될 수 있다. 문헌에서는 전사적으로 10여개[Jacobson et al., 1995] 또는 30~50개로 아직 정설이 없는 것으로 보인다. 한편 기능점수와 같은 방식을 공수산정에 도입한 경우, 유스 케이스는 약 200시간 정도의 공수가 투입되는 크기로, 기능 10여개, 또는 최대 기능점수가50점을 넘지 않도록 그 크기가 결정되기도 하였다. 특히 입력, 수정, 삭제, 조회가 포함된 이른바 CRUD 유형의 유스 케이스가 실무에서는 많이 인식되는데[Cockburn, 2001], 이 경우에도 적정한 크기가 되도록 인식하는 것이 요구사항의 관리와 구현에 도움이 된다.

〈표 9〉 유스 케이스의 적당한 화면 수

항 목	빈도	비율
1	1	3.7
2	9	33.3
3	13	48.1
5	1	3.7
6	3	11.1
총 응답자 수	27	100.0

4.3 유스 케이스 명세에 관한 인식

유스 케이스는 기존의 객체지향 분석설계 방법론(예 : OOSE, OMT)에서 필수적인 산출물로서 실무에 적용되고 있음에도 불구하고, 본 설문의 결과는 그 필요성을 강하게 뒷받침하지 못하였다. <표 10>에서 보듯이 응답자는 '도출된 모든 유스 케이스에 대해 유스 케이스 명세를 반드시 작성해야 한다'는 설문에는 비록 긍정적이었지만(2.67), 크게 긍정하는 것은 아니었다. 이러한 태도에 관해, RUP 에서는 유스 케이스가 서로 상속의 관계로서 연관되어 있는 경우, 그 내용이 매우 유사하면 유스 케이스 명세를 합해서 기술 할 수 있다고 하였다. 본 설문의 결과는 전사적인 프로젝트에서 데이터 중심의 입력, 수정, 삭제, 조회와 같은 유스 케이스가 많이 인식되는데 이 중 조회는 업무로직이 없고 단순하게 현재 데이터베이스를 있는 그대로 보여주는 경우가 많이 있다. 이러한 '단순조회'는 유스 케이스 명세를 하나하나 작성하는 노력에 비해 그 유용성이 적다고 판단되면 일반화로 처리해서 간단히 처리할 수 있는 관행을 반영한 듯하다 (예 : 연수자 목록 조회, 연수과목 목록 조회, 상품 목록 조회). 이러한 경우 유스 케이스 실현(realizations)을 활용하면 요구사항 추적성을 확보할 수 있다.

<표 10>에서 나타난 바와 같이 설문의 응답

자는 '유스 케이스가 요구사항을 담은 문서'로서 유용하다는 태도를 보이고 있지만, 그 정도가 기대보다 높지 않았고(2.78), 따라서 '유스 케이스 명세가 요구사항 정의서보다 요구사항을 반영하는데 더 유용하다'는 항목에 대해서도 매우 긍정적이지는 않았다(2.74). Simons et al. [2001]은 유스 케이스 모델링이 요구사항을 문서화하는데 있어 종전의 방식에 비해 난이도가 높다고 지적하고 있다. 또한 데이터흐름도(DFD)와 같이 요구사항을 계층화 해서 문제를 해결하는 방식에 익숙했던 사람들은 유스 케이스 모델링의 평면적인 구조에 대한 유용성에 의문을 제기할 수 있다[Regnell et al., 1996]. 유스 케이스 명세에서 응답자가 가장 어렵다고 느끼는 항목은 <표 11>에서 보듯이 대안흐름(alternative flow)이었다. 다음으로 특별요건(special requirement), 그리고 사후조건(post-condition) 순서로 나타났다. 유스 케이스 명세에서 대안흐름은 예외흐름과의 차이를 어려워했으며, 예외흐름을 Java의 Exception과 비교하는 경향이 많았다. 또한 유스 케이스 명세를 읽어본 결과 기본흐름(basic flow)을 제외하고 특별요건, 사전조건(pre-condition), 사후조건, 대안흐름을 충분하게 적지 않는 경향이 많았다. 이 중 사전, 사후조건은 유스 케이스의 기능적 요구사항으로서 클래스에 책임을 할당하는 도구로서 유용하게 활용될 수 있다[Meyer, 1997].

유스 케이스 명세는 자세한 정도에 따라 (1) 개요, (2) 분석 수준, (3) 설계 수준으로 나누는 것이 보통이다[Larman, 2002]. 분석 유스 케이스 명세는 사용자의 요구사항을 기술하는 데 반해, 설계 유스 케이스 명세는 요구사항이 어떻게 구현 되는지에 관해 기술한다. 그러나 설계 유스 케이스 명세에 어떤 내용을 적을 것인지에 대해서는 아직 일치된 견해가 없다. 본 설문의 결과는 유스케이스 명세가 분석보다는 설계 수준으로 작성되는 것이 좋겠다는 태도를 보여주고 있으며, 구현에 관련된 내용이 상세하게 포함되도록 업무규칙(2.48), 데이터베이스(2.63), 화면(2.78)이 충분히 기술되어야 한다는 점에 긍적적 태도를 보였다. 이는 <표 12>의 결과와도 일치하는데, 유스 케이스 명세에 업무규칙을 포함하는 것이 매우 중요하다고 응답했으며 (넓은 의미에서 계산식도 업무규칙으로 분류할 수 있음), 테이블 이름과 속성과 같은 데이터베이스에 관해 구체적인 내용이 유스 케이스 명세에 기술되기를 원했다. 그러나 데이터베이스 또는 화면이 변경될 때마다 유스 케이스 명세가 수정되어야 한다면 유지보수성이 취약해질 수 밖에 없다는 점에 유의해서 물리적인 내용을 가급적 피하고, 논리적으로 작성하는 것이 바람직하다. 예를 들어, '로그인 버튼 클릭'은 보다 물리적이고 이를 논리적으로 표현하면 '로그인 기능 선택'이 된다. 이는 본 설문의 결과와도 일치하는데, <표 10>의 '유스 케이스 명세가 화면이 바뀔 때 마다 재작성 되어야 하는가' 라는 질문에는 응답자는 부정적인 태도를 보였다(3.26).

위와 같이 유스 케이스 명세가 설계수준으로 작성되는 것이 좋겠다는 태도에도 불구하고, 응답자는 유스 케이스 명세를 구현할 때 참조는 하지만(2.78) 그다지 도움이 되지 않는다고 응답하였다(3.11). 이는 앞서 설명한 바와 같이 유스 케이스가 요구사항을 기술하는 데도 매우 유용하지도 않을 뿐만 아니라, 개발할 때도 매우 도움이 되지 않는다고 느끼는 점에서 흥미 있는 결과이다. 한 걸음 더 나아가, 응답자는 유스 케이스 명세에서 클래스를 도출하는 것이 쉽지 않다는 태도를 보였다(2.30). 이러한 태도는 객체지향 언어를 사용하지만 데이터베이스는 아직 관계형이 적용되고 있고, 따라서 테이블을 중심으로 클래스를 인식하는[Fowler, 2003] 보다 안정적인 방식을 택하려는 현상을 반영하고 있다.

〈표 10〉 유스 케이스에 관한 인식

	평균	표준편차
도출되는 모든 유스 케이스에 대해 유스 케이스 명세를 반드시 작성한다.	2.67	0.83
유스 케이스명세는 업무규칙에 관한 충분한 내용을 담는다.	2.48	0.70
유스 케이스명세는 데이터베이스에 관한 충분한 내용을 담는다.	2.63	0.84
유스 케이스명세는 화면에 관한 충분한 내용을 담는다.	2.78	0.89
유스 케이스명세는 요구사항을 담은 문서이다.	2.78	0.85
유스 케이스명세는 요구사항정의서보다 요구사항을 반영하는 도구로 더 좋다.	2.74	1.06
유스 케이스명세에 적힌 내용은 구현할 때 도움이 된다.	3.11	0.85
유스 케이스명세는 개발하면서 참조해야 하는 문서이다.	2.78	1.01
유스 케이스명세에서 클래스를 도출하는 것은 어렵다.	2.30	0.82
유스 케이스명세는 매 단계별로 반복할 때 마다 개정한다.	2.92*	1.06
유스 케이스명세는 화면이 바뀔 때 마다 재작성 한다.	3.26	0.98
유스 케이스명세는 테스트할 때 참조해야 하는 문서이다.	3.00	0.92
유스 케이스명세서는 인수에 필요한 문서이다.	2.50*	0.95

주) 별표(*) 항목 : N = 26, 나머지 항목 : N = 27

〈표 11〉 유스 케이스 명세에서 작성 난이도가 높은 항목

항 목	빈 도	비 율
BASIC FLOWS	3	11.1
ALTERNATIVE FLOWS	16	59.3
SPECIAL REQUIREMENTS	7	25.9
PRECONDITION	3	11.1
POSTCONDITION	4	14.8
총 응답자 수	27	복수 응답 가능

〈표 12〉 유스 케이스 명세에 기술할 내용

항 목	빈 도	비 율
테이블 이름과 속성	18	66.7
업무 규칙	24	88.9
계산 식	10	37.0
화면	7	25.9
총 응답자 수	27	복수 응답 가능

<표 10>의 '유스 케이스 명세를 매 단계별로 반복할 때 마다 개정하는가' 라는 질문은 응답자가 요구사항에 대해 일시에 상세화하는 폭포수적인 태도를 보이는지 또는 단계별로 진화하는 반복 점진적인 태도를 보이는지 알아보기 위한 항목이었다. 설문 결과는 유스 케이스 명세의 반복적인 개정에 대해 긍정적이지만 그 정도가 중립에서 크게 벗어나지 않고 있음을 보여주고 있다(2.92). 이는 사용자의 요구사항이 매우 가변적이고, 이를 보다 효과적으로 수렴하기 위한 대안으로 제시되고 있는 반복적 방식에 배치되는 입장을 역설적으로 보여주고 있다. Larman[2002]은 유스 케이스에 대하여 착수단계에서는 80% 인식, 20% 상세를, 상세단계에서는 80%의 상세가 바람직하며, 나머지 요구사항도 구현단계에서 보완될 수 있다고 지적하였다.

최근에는 테스트에 중점을 둔 방법이 관심을 끌고 있다[Martin, 2003]. 우량의 소프트웨어를 개발하려면 적절한 테스트가 항상 선행되어야 하는데, 특히 테스트케이스를 누가 언제 어떻게 작성할 것인지는 항상 논란의 소지가 있다. 예를 들어, 카드승인의 경우 테스트 케이스에 필요한 의사결정 테이블을 작성하려면 유스 케이스 당 1000개 이상이 필요하며, 모든 경우의 수에 대해 실제 데이터를 사용하여 이를 작성하는 데만 10여 시간 이상이 걸리게 된다. 이와 같이 실무에서 테스트 케이스의 유용성에 대한 태도를 반영하여, 본 설문에서 응답자는 유스 케이

스 명세가 테스트에 유용하다고 생각하지 않고 있었다(3.00). 이 보다는 유스 케이스 명세가 요구사항을 기술한 문서로서 모든 시나리오가 적절히 구현되었는지 확인하는 인수문서로서의 유용성을 인정하고 있는 태도를 보여주었다(2.50).

〈표 13〉 유스 케이스명세의 작성자

구 현	빈 도	비 율
개발자	2	7.4
현업사용자	4	14.8
분석/설계 담당자	4	14.8
현업사용자와 개발자가 같이	6	22.2
현업사용자와 분석/설계 담당자가 같이	15	55.6
개발자와 분석/설계 담당자가 같이	1	3.7
총 응답자 수	27	복수 응답 가능

'한편 유스 케이스 명세를 누가 작성해야 좋은가' 라는 질문에 대해, 응답자의 55.6%가 현업사용자와 분석/설계 담당자가 같이 하는 것이 좋다고 응답했다(<표 13> 참조). 유스 케이스 명세는 요구사항이 기술된다는 점에서 현업사용자가, 이러한 요구사항을 설계에 반영해야 한다는 점에서 분석/설계 담당자가 같이 참여하는 것이 바람직하다. 흥미 있는 결과는 현업사용자가 개발자와 같이 유스 케이스 명세를 작성해도 좋겠다는 의견이 22.2%나 되었다는 점이다. 이는 '분석설계' 작업에 대한 의문을 제기한 것으로, 이럴 경우 많은 문제점이 지적될 수 있다. 우선, (1) 순공학과 역공학이 병행하기 보다는 역공학이 우선하게 되어서, 분석설계의 통찰력이 반영될 기회를 잃을 수 있게 된다. 또한 (2) 현업사용자의 요구사항이 개발편의성과 타협하면서 품질에 악영향을 미칠 수 있다. 마지막으로, (3) 분석설계자와 개발자의 상호검토에 따른 순기능이 없어질 수 있게 되는데. 업무를 분장하여 분석설계 담당자가 유스 케이스를 작성하면 내부검토가 활성화되면서 품질을 향상시킬 수 있다. 즉, 개발자와 분석설계사가 서로의 잘못된 점을 지적하며, 활발한 검토의 기회를 제공하는 것이다.

5. 토론 및 결론

본 연구에서는 유스 케이스 모델링에 관한 활용실태를 조사하였는데, 그 결과 유스 케이스는 많은 객체지향적 방법론에서 채택되고 있음에도 불구하고 여전히 어려움을 수반하고 있는 것으로 나타났다. 우선 유스 케이스 모델링이 인지적으로 어렵다는 점은 여러 문헌에서도 지적되고 있는 바와 같이[Cockburn, 2001 ; Cox, 2000] 본 논문의 결과도 이러한 주장을 뒷받침하고 있다. 본 연구는 실무에서 유스 케이스가 기능보다 인식하는 것이 어렵고 또한 요구사항 문서로서의 유용성에 대해서도 회의적인 시각을 잘 보여주고 있다. 이러한 회의적인 인식은 객체지향이 기능적 방식에 비해 난이도가 높다는 점과 상승작용을 일으키면서 객체지향의 장점에도 불구하고 그 확산에 제동을 걸 수 있다는 점에서 주의가 필요하다. 그 이유로서 본 연구의 응답자들이 폭포수 모형에 익숙해 있어서 과거로 회귀하려는 경향으로 인해 그러한 응답을 했다고 추정할 수 있다. 그러나 본 연구의 응답자는 객체지향에 대해 비교적 잘 알고 있을 뿐만 아니라, 과거에 폭포수방식의 프로젝트를 수행한 경험이 그리 많지 않다는 점을 감안하면 앞의 추정은 적절한 추정이라고 할 수 없다. 그렇다면 기존 문헌에서 지적되고 있는 이러한 UML에 내재하고 있는 인지적 어려움을 줄이기 위한 처방이 UML에 반영될 필요가 있다. 또한 기능적인 사고에 익숙한 점을 감안하여, 상향식

유스 케이스 모델링을 시도하는 것도 인지적 어려움을 줄이기 위한 방안으로 시도될 수 있다[Regnell et al., 1996]. 즉, RUP의 절차에 따라 행위자와 목적을 먼저 인식하고, 유스 케이스를 하향식으로 인식하기 보다는 사용자의 이벤트와 시나리오를 인식하고 유스 케이스를 인식하는 상향식 방식이 유용하게 활용될 수 있다. 여기서 기능점수에서 정의하고 있는 EI, EO, EQ와 같은 기능과 점수를 모델링에 활용하여 내용을 기술하거나 유스 케이스의 크기를 통제할 수도 있다.

또한 유스케이스 명세는 많은 내용을 포함하면서도, 정작 시스템을 구현하면서 실무에 활용하기에는 부족하다는 점에 주목할 필요가 있다. 그 이유로서, 유스 케이스 명세를 작성한 후 바로 프로그래밍과 같은 개발이 뒤 따르는 것이 아니라, 그 중간 과정에 (1) 클래스 인식, (2) 책임 분할, (3) 데이터 모델링, (4) 클래스 메소드에 대한 명세를 작성하는 과정이 있고, 이들 산출물이 모두 구현 작업을 하는데 활용되기 때문이다. 특히 클래스 메소드에 대한 명세는 과거 프로그램 명세와 같이 의사코드를 사용하여 기술하면 더욱 유용하게 활용될 수 있다. 그렇다면 유스 케이스 명세가 많은 객체지향적 개발방법론에서 강조하는 만큼 유용하게 활용될 수 있는 방안은 무엇일까? 이와 관련하여 유스 케이스 명세를 좀 더 효과적으로 활용하기 위한 대안이 몇 몇 연구에서 제시되고 있다. 명세는 보통 자연언어로 표현하지만[Jacobson et al., 1992 ; Rumbaugh et al., 1991] 애매모호성으로 인한 한계를 극복하기 위해 구조화된 자연언어[Nurcan et al., 1998], 유사코드, 그래프 기반 표기법[Regnell et al., 1996] 등이 제시되고 있다. 뿐만 아니라, 본 연구의 결과는 유스 케이스 명세와 구현모델 간의 이행이 자연스럽지 못하고 적지 않은 혼란이 있다는 것을 시사하고 있는데, 구체적이고 유용한 실무적인 지침이 시급하다는 점을 웅변하고 있다. 특히 본 논문에서 지적된 문제점으로는 (1) 유스 케이스의 크기(granularity)가 매우 주관적이고, (2) 명세에서 클래스를 인식하는 것이 쉽지 않고, (3) 명세에 구현과 관련된 내용(예 : 데이터베이스, 화면)을 많이 포함하면서 유지보수성이 저해될 수 있고, (4) 명세에 적어야 할 항목 중 시나리오와 사전조건, 사후조건을 기술하는 것이 쉽지 않은 것으로 보여, 이에 대한 구체적이고 실무적인 지침이 필요하다. 문제점의 하나인 유스 케이스의 크기에 대해서는 계량화된 기준을 적용하는 것이 유용한데, 예를 들어 기능점수를 적용하여 임계값을 넘지 않도록 그 크기를 제한할 수 있다. 유스 케이스 명세에서 클래스를 인식하기 위해서는 언어학적으로 접근해서 명사는 클래스로, 동사는 클래스의 메소드로 인식할 수 있다(예, Liang [2003]). 또한 유스 케이스의 내용에 포함될 내용은 가급적 논리적인 구조로 유지보수성을 향상하는 것이 필요하며, 이는 패키지를 구조화하면서 발생할 수 있는 순환구조를 피하라고 하는 지침과도 상통한다[Martin, 2003].

위와 같은 유스 케이스에 대한 유용성에 대한 의문은 유스 케이스에 대한 이해가 부족한 것이 한 원인이 될 수 있다. 특히 유스 케이스가 요구사항을 반복적, 점진적으로 구체화 시켜나가는 산출물로서 활용하는 것이 중요함에도 불구하고, '폭포수 모형' 시각에서 이해하는 경향이 있었다.

유스 케이스 모델링을 도입한다는 것은 단순히 기술적인 변화이기 보다는 프로젝트 관리에도 큰 변화가 있다는 점을 인식해야 한다. 무엇보다 먼저, 객체지향 프로젝트에서 유스 케이스는 프로젝트의 전 과정에 걸쳐 효과적으로 관리하여 개발자로 하여금 유스 케이스를 부가적인 문서로 생각하는 경향을 경계해야 한다. 또한,

반복적 방법론을 사용하면 요구사항이 프로젝트 시작 전에 정확히 나올 수가 없기 때문에 프로젝트 비용을 산출하는 데 어려움이 있다는 점을 인식하고 프로젝트 초기인 착수단계에서 유스 케이스를 자세히 분석하기 보다는 모든 유스 케이스를 인식하도록 하는 것이 중요하다. 폭포수 방식에서는 사전에 잘 정의된 요구사항에 따라, 기능별로 점수를 내어 비용을 산정할 수 있지만, 유스 케이스는 기능절차가 묶인 것으로 기능적 방법에서와 같이 계층적으로 기능이 분할되지 않아 비용을 산출하기가 쉽지 않다. 그러나 국내에서의 관행은 프로젝트 계약이 사전에 이루어져야 하는 까닭에, 이러한 점을 극복하기 위해 유스 케이스의 크기를 비교적 일관된 크기로 결정하여 사전에 몇 개의 유스 케이스가 도출 될 것인지를 추정하는 것도 한 방법이 될 수 있다. 반복적 방법론에서 유스 케이스는 그 중요성에 비해 활용성이 낮게 평가될 수 있다. 마지막으로, 프로젝트의 일정은 유스 케이스 중심으로 T자 모양으로 시스템이 통합되고 확장되는 모습으로 반복계획을 작성하여 실질적으로 유스 케이스가 프로젝트의 중심이 되도록 해야 한다. 현재는 유스 케이스가 프로젝트의 중심으로 요구사항부터 구현까지 추적성을 제공하는 유용한 도구임에 틀림없으나 한국적 개발관행에 쉽게 적용될 수 있는 실질적인 가이드라인과 방법론의 제시가 시급하다.

유스 케이스는 그 서식이 기술식이고, 어느 정도 창의적인 사고를 요구한다는 점에서 한국적 풍토에는 적합하지 않은 면이 있다고 판단된다. 따라서, 유스 케이스 명세는 개발자보다는 별도로 분석설계 담당자를 할당하여 작성하게 하고, 개발자로 하여금 이를 참조하여 시스템 구현에 활용하는 방안이 바람직하다. 특히 유스 케이스를 도출할 때는 기능과 분명한 차이가 있다는 점을 인식하고 작업을 해야 하며, 유스 케이스가 잘못 도출되면 클래스의 재사용과 작업의 효율성이 저해되게 된다. 유스 케이스는 몇몇의 분석설계 담당자가 작업을 하여 그 크기가 주관적이고 계량화 되어 있지 못하는 단점이 있을 수 있으나 이러한 문제점을 극복해 일관된 크기로 결정될 수 있게 하는 것이 프로젝트 계획의 적정성과 실천성을 향상시킬 수 있다. 그렇지만 유스 케이스는 그 목적상 크기가 서로 다를 수 있다는 점을 인정해야 한다. 유스 케이스는 요구사항을 파악하는 문서로서 반복마다 개정됨을 원칙으로 한다. 따라서 단계별로 개정될 때 마다 그 상세화 정도도 진화되어, 버전관리가 필요하므로 각 유스 케이스에 구분자를 붙여 구분하는 것이 유용하다.

유스 케이스는 객체지향 개발에 있어 시작이요, 끝이라는 인식을 갖고 사용자 중심으로 시스템이 구현되도록 한다. 그러나 유스 케이스 또는 유스 케이스 명세 그 자체가 개발에 직접 활용될 수는 없다. 유스 케이스는 기능의 집합인 점을 감안하여, 기능별로 분석하여 클래스, 화면, 테이블이 도출되는 것이 바람직하다. 보통 개발자는 기존의 시스템이 있을 경우, 유스 케이스와 별도로 작업하여, 화면에 신규 요구사항을 받고, 테이블을 도출하여 개발하는 경향이 있으나, 이는 유스 케이스의 추적성을 해치는 결과를 초래할 뿐 아니라, 사용자의 프로젝트가 종료될 때까지 요구사항이 불안정해지는 결과를 낳을 수 있다. 유스 케이스는 상세화 되면서 화면에 종속되는 경향이 있으나, 분석단계는 철저히 기능중심으로 명세를 작성하여, 독립성을 유지하도록 한다. 설계 유스 케이스 명세도 가급적 화면의 물리적인 면이 배제되도록 하여 화면이 바뀔 때마다 유스 케이스가 갱신되어 관리되는 번거로움을 최소화 하도록 한다. 유스 케이스 명세는 시나리오를 담고, 각 시나리오 별로 테스트 케이스가 작성되어, 유스 케이스 그

자체가 인수의 대상이 되기 보다는 테스트 케이스에 담긴 시나리오와 화면을 대상으로 인수하게 하고, 다음에 이들을 묶은 유스 케이스가 인수되도록 하는 것이 바람직하다.

본 논문의 결과는 설문의 응답자가 적다는 점에 주의해서 이해해야 하며, 향후유용한 유스 케이스 지침이 만들어지고 이에 따라 유스 케이스가 작성될 경우, 이러한 인식에 어떠한 차이가 있는지를 충분한 표본을 대상으로 다시 살펴보는 것도 의미 있는 작업이라 하겠다.

참 고 문 헌

[1] Cockburn, A., *Writing Effective Use Cases*, Addison-Wesley, 2001.

[2] Cox, K., "Cognitive Dimensions of Use Cases Feedback from a Student Questionnaire," *12th Workshop of the Psychology of Programming Interest Group*, Cozenz Italy, April 2000, pp. 99-122.

[3] Fowler, M., *Patterns of Enterprise Application Architecture*, Addison-Wesley, Sydney, 2003.

[4] Iivari, J. & Maansaari, J., "The Usage of Systems Development Methods : Are We Stuck to Old Practices?," *Information and Software Technology*, Vol. 40, 1998, pp. 501-510.

[5] Jacobson, I., Christerson, M., Jonsson, P. & Overgaard, G., *Object-Oriented Software Engineering : A Use Case Driven Approach*, Addison-Wesley, 1992.

[6] Jacobson, I., Ericsson, M. & Jacobson, A., *The Object Advantage : Business Process Reengineering with Object Technology*, Addison-Wesley, 1995.

[7] Larman, C., *Applying UML and Patterns An Introduction to Object-Oriented Analysis and Design and the Unified Process*, 2nd Ed., Prentice-Hall, 2002.

[8] Leffingwell, D. & Widrig, D., *Managing Software Requirements : A Unified Approach*, Reading, MA : Addison-Wesley, 2000.

[9] Lindvall, M., *A Study of Traceability in Object-Oriented Systems Development*, Licentiate Thesis 462, Dep. Of Computer and Information Science, Linkping University, Sweden, 1994.

[10] Liang, Y., "From Use Cases to Classes : A Way of Building Object Model with UML," *Information and Software Technology*, Vol. 45, 2003, pp. 83-93.

[11] Martin, R. C., *Agile Software Development Principles, Patterns, and Practices*, Prentice-Hall, 2003.

[12] Meyer, B., *Object-Oriented Construction, 2nd Ed.*, Upper Saddle River, NJ : Prentice-Hall, 1997.

[13] Nurcan, S., Grosz, G., Souveyet. C., "Describing Business Processes with a Guided Use Case Approach," *Lecture Notes in Computer Science 1413*, Pernici, C., Thanos(Eds.), Springer, Pisa, Italy, June 1998.

[14] Regnell, B., Anderson, M. & Bergstrand, J., "A Hierarchical Use Case Model with Graphical Representation," *Proceedings of ECBS'96, IEEE International Symposium and Workshop on Engineering of Computer-Based Systems*, 1996.

[15] Rumbaugh, J., Blaha, J., Premerlani, W., Eddy, F. & Lorensen, W., *Object-Oriented Modeling and Design*, Prentice-Hall, 1991.

[16] Sheetz, S.D. & Tegarden, D.P., "Illustrating the Cognitive Consequences of Object-Oriented System Development," *The Journal of Systems and Software*, Vol. 59, 2001, pp. 163-179.

[17] Simons, J., Graham, K., Owen, A., Patterson, K., and Hodges, J., "Perceptual and Semantic Components of Memory for Objects and Faces : A PET Study," *Journal of Cognitive Neuroscience*, Vol. 13, No. 4, 2001, pp. 430-443.

저자소개

정 승 렬

미국 사우스 케롤라이나 대학에서 경영정보학 박사를 취득하고 현재 국민대학교 비즈니스IT전문대학원 교수로 재직중이다. 그는 ERP 시스템을 포함하여 다수의 공공분야 및 금융권 대형 시스템 개발 프로젝트에 참여하였으며 유명 국내외 저널에 시스템 개발 및 구현, 정보시스템 감리, ERP, process management 등의 주제와 관련하여 많은 논문을 발표하였다.

임 좌 상

호주 뉴사우스웨일즈 대학에서 정보시스템 박사를 취득하고 현재 상명대학교 미디어학부 교수로 재직중이다. 그는 객체지향/컴포넌트 개발, 소프트웨어 품질공학 및 감성공학 등의 주제와 관련하여 컨설팅 및 개발 프로젝트를 수행하였으며 국내외 유명 학술지에 다수의 논문을 발표하였다.

◈ 이 논문은 2004년 4월 28일 접수하여 1차 수정을 거쳐 2004년 10월 6일 게재확정되었습니다.