객체지향기술: '소프트웨어 위기'를 해결하기 위한 해결책(Ⅱ)

(Object-Oriented Technology: A Silver Bullet for Software Crisis)

임성택/고려대학교
경상대학 경영정보학과 조교수
Rim, seong-taek. ph.D. Assitant Professor,
Dept Of Management Information Systems College Of Economics & Commerce. Korea University.

연재순서

1. 서론
2. 객체지향의 기본개념
3. 객체지향 소프트웨어공학
4. 객체지향 데이터베이스
5. 결론

3. 객체지향 소프트웨어공학 (Object-Oriented Software Engineering)

사용자들이 원하는 시스템의 규모가 갈수록 커지고 복잡해짐에 따라 이들 시스템을 개발하기 위해서는 종래의 소프트웨어 개발방법과는 아주 근본적으로 다른 새로운 소프트웨어 개발방법이 필요하게 되었다.

전통적인 시스템개발방법은 시스템 개발의 각 단계가 반복되지 않고, 단계별 구분이 너무 명확하여 전체적인 통합이 어렵고, 재사용을 강조하지 않는다등 여러가지 문제점을 내포하고 있다. 객체지향 소프트웨어공학은 이러한 문제점의 대부분을 해결해 줄 혁명적인 소프트웨어 개발방법으로 여겨지고 있다. 벌써 일부 사람들은 객체지향방법론을 그들이 신봉해야할 새로운 종교로 받아들이고 있다.

하지만 1970년대와 1980년대에 구조적 방법론도 소프트웨어 위기를 극복할 수 있는 방안으로 많은 사람들의 환영을 받았었다. 1980년대 중반 CASE (Computer Aided Software Engineering)도 마찬가지였다. 이제는 객체지향기술이 1990년대의 십자군인 셈이다. 과연 객체지향방법론이 소프트웨어의 위기를 해결할 수 있는 해결책(silver bullet)인지 아니면 이전의 다른 방법론들처럼 또하나의 새로운 미봉책에 불과할 것인지에 대해 진지하게 생각해 볼

필요가 있다.

이번 절에서는 소프트웨어의 산업혁명으로 일컬어지는 객체지향 소프트웨어공학에 대해 소개하려 한다. 여기서는 객체지향 소프트웨어공학이란 무엇이며, 구조적 방법론에 비교해 볼 때 어떤 장단점이 있는지를 알아보자.

3.1 객체지향 소프트웨어 공학

3.1.1 소프트웨어의 산업혁명

객체지향방법에 의한 소프트웨어의 개발은 소프트웨어의 산업혁명에 비유될 수 있다. 즉 산업혁명시의 제조업에서 일어난 변화는 소프트웨어 개발시에 객체지향방법을 적용할 때 일어나는 변화와 아주 유사하다. 200여년 전에는 오늘날과 우리가 아는 것과 같은 제조업은 없었다. 하나의 제품은 오랜 도제관계를 통해서 숙련이 된 고도의 기술을 가진 장인에 의해서 한번에 하나씩 만들어졌다. 각 장인은 제품생산의 처음부터 끝까지 책임이 있었다. 실제로 각 제품은 하나의 예술품이었다. 하지만 산업혁명후에는 각 부분품의 전문가가 각기 맡은 부분만을 생산하고, 최종상품은 이들 부분품의 조립에 의해서 만들어졌다. 이는 제품생산에 있어서 패러다임의 전환이라고 할 수 있다.

마찬가지로 전통적인 방법에 의한 소프트웨어의 개발은 200여년전의 제조업에 비유할 수 있다. 이렇게 해서 만든 소프트웨어는 아주 비싸고, 만드는 데 많은 시간이 소요되고, 품질은 개발자 개개인의 능력에 달려있다. 또한 각 소프트웨어는 독특하기 때문에 표준화하기가 쉽지않고 유지보수도 어렵다. 하지만 객체지향방법에 의한 소프트웨어의 생산은 전문화된 업자들이 의한 클래스를 개발하고 최종 소프트웨어(프로그램)은 이들 클래스의 조합에 의해서 만들어진다. 이는 소프트웨어 개발에 있어서 패러다임의 전환이라고 말할 수 있다.

3.1.2 객체지향 소프트웨어의 구성요소

객체지향기술는 맨먼저 객체지향프로그래밍에서 시작하여 객체지향설계를 거쳐, 현재 객체지향분석까지 발전해 왔다. 객체지향개념이 소프트웨어 생명주기 전단계에 적용된다는 의미에서 객체지향 소프트웨어공학이란 용어가 사용되고 있다. 따라서 객체지향 소프트웨어공학이란 크게 객체지향분석, 객체지향설계, 객체지향프로그래밍의 세단계로 구분해 볼 수 있다. 단 한가지 언급하고 싶은 것은 객체지향분석이나 설계의 구분이 객체지향 소프트웨어공학에서는 명확하지 않다는 것이다.

● 객체지향분석

시스템 분석이란 시스템이 해야할 일을 설명할 목적으로 문제영역(실세계)을 이해하고, 이를 모델링하는 것이라고 할 수 있다. 따라서 객체지향분석이란 객체의 관점에서 시스템의 요구사항을 분석하고 모델링하는 과정이라고 할 수 있다. 다시말해 객체, 클래스, 속성, 메소드등을 찾아내고 그들간의 관계나 행위를 규정하는 것이다. 대표적인 방법론으로는 Coad와 Yourdon, Rumbaugh, Shaler 와 Mellor의 방법론을 들 수 있다.

● 객체지향설계

시스템 설계란 분석모델을 좀 더 발전시켜 컴퓨터 구현을 위한 좀 더 상세한 기초를 제공하는 것이라고 할 수 있다.

이는 문제영역(problem domain)에서 해결영역(solution domain)으로 바뀐다는 것을 의미한다.

하지만 여기서는 시스템이 어떻게 작동할 수 있을 것인지를 설명하는 것이지 구체적인 구현 방법에 대해서 논의하는 것은 아니다. 객체지향 설계란 객체를 분석의 대상으로 생각하고 해결 영역을 모델링하는 것이다. 대표적인 예로는 Booch, Wirfs-Brock, Shlaer와 Mellor의 방법론이 있다.

● 객체지향프로그래밍

객체지향프로그래밍은 프로그램이 객체들의 집합으로 이루어지는 구현방법이다. 여기서 객체란 어떤 클래스의 인스탄스이고 그 클래스들은 상속 구조로 조직되어 있다. 원래 객체지향 개념은 객체지향 프로그래밍에서 주로 출발한 것이며, 그 후로 객체지향설계, 객체지향분석이란 개념으로 발전하였다. 객체지향 프로그래밍에서 객체지향 프로그래밍의 기본 개념에 대해서는 이미 객체지향기술의 개념부분에서 살펴보았다.

3.2 객체지향 소프트웨어 공학의 장점

객체지향 소프트웨어 공학의 가장 큰 장점은 재사용에 있다. 물론 이러한 장점은 재사용할 가치가 있는 클래스가 있어야 하고, 그러한 클래스들은 쉽게 통합될 수 있도록 표준화되어 있음을 전제로 한다. 객체지향 소프트웨어공학에서는 소프트웨어 개발을 "확장에 의한 설계(design by extension)"란 개념으로 생각한다. 다시말해 여기서는 새로운 시스템을 이미 개발된 시스템의 확장으로 생각한다. 하지만 구조적 방법론을 사용하는 사람들은 매 프로젝트마다 전에는 경험하지 못한 새로운 것으로 생각한다.

객체지향 소프트웨어공학의 두번째 장점은 데이터와 기능을 한꺼번에 분석하고 표현할 수 있다는 것이다.

객체지향 소프트웨어 공학에서는 데이터와 기능을 객체라는 한 용기에 담는다. 더우기 이들 데이터와 기능들은 하나의 다이아그램에서 함께 표현될 수 있다. 구조적 방법론에서는 기능의 모델링을 많이 강조하고, 데이터의 모델링에는 그리 큰 관심을 기울이지 않는다. 물론 최근에 와서 E-R 다이아그램을 구조적 방법론의 일부로 영입했지만, 아직도 많은 개발자들이 사용자의 요구사항을 분석하기 위해 대부분의 에너지를 데이터 흐름도를 그리는데 소비하고 있다.

객체지향 소프트웨어공학은 그래픽 사용자 인터페이스를 위한 클래스 라이브러리를 제공한다는 것이다. 구조적 방법론은 사용자 인터페이스를 개발하는데 아무런 가이드를 제공해주지 않았다. 이것은 대부분의 구조적 방법론이 사용자 인터페이스가 별로 중요시되지 않은 1970년대에 개발되었기 때문이다. 그당시 대부분의 입력수단은 터미날에서의 문자였으며, 출력도 마찬가지로 단순하였다. 하지만 오늘날의 상황은 아주 다르다. 윈도우, 마우스, 아이콘 환경하에서는 프로그램의 75%이상이 사용자 인터페이스와 관련되어 있다.

3.3 객체지향 소프트웨어공학의 문제점

객체지향 소프트웨어공학이 앞에서 언급한 여러가지 장점이 있는 것도 사실이지만 한편으로 문제점이 없는 것은 아니다. 물론 이러한 걱정이 대부분 일시적이고 앞으로 객체지향관련 연구와 시장이 성숙됨에 따라 대부분 해소될 것이지만 지금 당장 객체지향 소프트웨어공학방식을 도입하려는 사람들은 당연히 고려해 볼 사항이다.

첫째, 객체지향기술의 성숙도이다. 소프트웨

어 개발에 객체지향방법을 적용하는 것은 아직 시기상조인 것같다. 또한 대부분의 시스템들이 아직 실험실 단계에서 행하여졌으며, 더욱 중요한 것은 객체지향방법에 의해 소프트웨어를 개발하는 것이 아직은 과학(science)이라기 보다는 예술(art)에 가깝다는 것이다. 다시말해 구조적 방법론에서 처럼 소프트웨어 개발을 가이드해 줄 정형화되고 증명된 소프트웨어 개발방법론(methodology)이 아직은 없다는 것이다.

둘째, 표준화에 대한 필요성이다. 아직 표준화가 덜 되어 있다는 것은 여러 소프트웨어 업자들이 개발한 클래스들을 혼용해서 쓰기가 힘들다는 것을 의미한다. 물론 아직은 클래스를 개발하는 소프트웨어 업체가 소수이기 때문에 별 문제가 아니라고 할 수 있지만 가까운 장래에 이는 심각한 문제가 될 것이다.

셋째, 좀 더 좋은 개발도구가 필요하다는 것이다. 구조적인 방법론과 비교해 볼 때 개발노력을 지원해줄 개발도구가 충분하지 않다. 새로운 기술을 지원하는 도구는 항상 새로운 기술에 뒤떨어져 개발된다. 객체지향 소프트웨어공학도 예외는 아니다.

넷째, 자격이 있는 인원을 충원하는 것이 문제이다. 다시말해 객체지향 소프트웨어공학을 효율적으로 사용할 수 있는 사람이 아직은 적다는 것이다. DC

Abstract

In recent years an OOT(Object-Oriented Technology) has received tremendous attention in analyzing, designing, and programming complex software systems. Although the OOT has become popular, there has been much confusion and controversy about what object-oriented means. The purpose of this paper is to introduce OOT as a silver bullet for software crisis to corporate managers. Section I discusses three basic concepts of OOT - i.e., objects, message, and classes. Section II introduces the object-oriented software engineering. Section III introduces the object-oriented database.