객체지향 환경과 EDP 감사기법의 응용

曺基祚
경남대학교 경영학부

요약

RDB나 기존의 ISAM 파일이 검색과 저장의 효율성이 높을지라도 다양한 형태의 자료 저장 능력과 프로그래밍의 효율성 부족에 대한 대안으로 객체지향 프로그래밍과 객체지향 DB의 활용이 확대되고 있는데 1996년, 미국의 경우 활용기업의 비율이 15%정도에 이르며 증가추세에 있는 것으로 나타났다. 본 연구는 OOP, OODB를 EDP 감사의 관점에서 연구하는 것으로 전통적 EDP 감사도구와 기법을 객체지향모형에 적용가능한지와 적용시에 예상되는 문제점을 검토하려는 것이다.

주요한 EDP 감사도구와 기법중 입출력에 초점을 두는 Test Data, Integrated Test Data/Facility에는 적용이 비교적 쉽지만 처리과정에 초점을 두는 Embeded Audit Module, GAS 등에는 구현을 위한 메소드의 수정이나 설치상의 문제가 큰 것으로 보인다. 그러므로 GAS를 구현할 때 감사모듈을 설치하는 절충법이 필요한 것으로 보인다.

제 1 장 서론

회계시스템의 설계와 구현을 현저히 변화시키는 계기가 있는데 그 예로는 관계형 DB의 수용, 사상지향 모형, PC의 활용, 통신망의 접속 등이 있다. 90년대에 들어서 일어난 변화의 한 예로 객체지향 모형이 개발되었는데 이는 구조적 방법의 도입이래 가장 중요한 시스템 개발방법이라 할 수 있다.(Ambler 1995)

RDB나 ISAM 방식의 파일구조는 저장과 검색의 효율성을 제고하였지만 텍스트 중심의 데이타에 한정된 저장능력 때문에 대체적 방법이 모색되어 왔다. 실제로 RDB나 ISAM 방식의 파일에 저장되어 있는 자료를 검색하는데에 객체지향 DBMS가 사용되고 있으며 객체지향 DBMS는 다양한 포맷의 자료 파일을 통일된 DB로 통합하는데에도 쓰이고 있다.(Romney et al. 1997)

객체지향 기법은 시스템 분석과 설계단계뿐만 아니라 프로그래밍 단계에서도 그 능력을 발휘한다. 이 경우마다 객체지향 방법은 자료가 저장되는 방법과 자료를 취급하는 방법을 지원할 자료구조를 재 정의한다. 이것은 유력한 정보시스템의 패러다임이 된다.(Adamson and Dilts 1995)

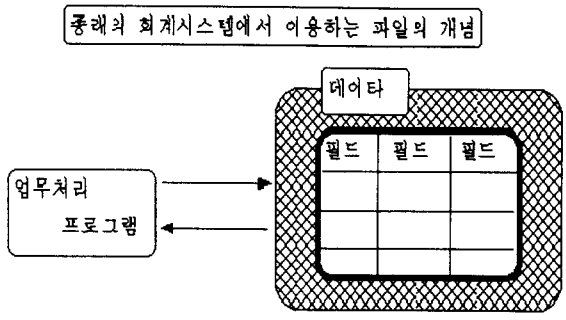
객체지향 자료모형이 회계시스템의 기본이 되므로 중요한 문제는 전통적 시스템의 회계감사를 위하여 개발된 기법을 어떻게 하면 새로운 회계시스템의 감사에 적용할 수 있도록 하는가 하는 점이다. 이 논문은 이 문제를 개념적 수준에서 다룬다. 우선 객체지향 모형을 서술하고 회계시스템의 설계방법에 어떤 차이가 있는가를 확인코자 한다.

제 2 장 OODM의 이해

1. 전통적 방법과 OODM의 출현

회계시스템은 적절한 파일을 설계하여 만들었고 그 자료를 조작하고 접근하기 위하여 업무처리 프로그램을 만들었다. 예를 들면 작업시간카드는 급여파일에 입력하고 급여처리 프로그램이 파일에 접근하여 급여를 계산한다. 그림 1에서 프로그램 코드와 자료는 독립적인 개념인데 데이터가 수동적인데 비하여 코드는 능동적이다. 이 기본적 모형은 데이터를 RDB에 저장하고 SQL 등과 같은 DML이 자료를 조작할 때와 같다.

전통적 모형은 세상을 행위가 수행되는 자료라고 본다. Coad and Yourdon(1991)은 이 전통적 모형의 문제가 자연스럽지 못하다는 것을 지적하였다. 오히려 세상을 서로간에 교류



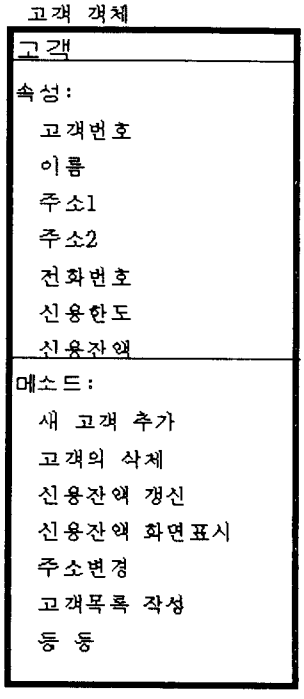
(그림 1) 종래의 회계시스템에 이용되는 파일의 개념

함으로서 상호작용 하는 사물과 사건의 집합으로 본다. 예를 들면 고객이 회사에 상품을 주문하거나 회사가 매입처에 매입채무를 갚기 위하여 수표를 보내거나 비디오 가게가 고객에게 테이프를 빌려주는 것 등이다. 나아가 사물을 유사성에 근거하여 분류함으로서 우리의 환경에 계층구조를 적용한다. 예를 들면 투자 포트폴리오는 유가증권과 부동산으로 구성되고 유가증권은 또 주식과 채권으로, 주식은 성장주나 수익주, 채권은 국채와 지방채, 회사채 등으로 분류될 수 있을 것이다. 고객은 중개인에게 부탁하여 유가증권을 더 구매할 수도 있을 것이다. 객체지향은 이러한 실상과 상호작용을 전통적인 방법보다도 더 잘 모형화할 수 있는 도구와 프로그래밍언어를 제공한다.

2. 객체자료모형

객체지향모형에서 세상은 객체라 부르는 실체들로 구성되어 있고 객체는 클래스계층에서 분류된다. 객체는 서로간에 감추어진 메소드나 각 객체내부에 캡슐화 된 메시지를 보냄으로서 서로 관계하거나 소통한다.(Murthy and Wiggins 1993) 이러한 내용을 상술하고자 한다.

Martin and Odell(1990)은 객체를 우리가 자료를 저장하고 그 자료를 조작하는 방법에 관한 실질적이거나 추상적인 일이라고 하였다. 예를 들면 앞에서 말한 고객의 유가증권은 고객 객체, 유가증권 객체, 중개인 객체가 될 것이다. 객체지향을 유일하고 강력하게 만드는 것은 객체가 객체에 있는 자료를 조작하기 위한 메소드를 포함하고 있다는 것이다. 고객 객체의 예에서 각 고객에 대한 고객번호, 이름, 주소, 신용한도, 외상잔고 등등 많은 속성을 보게된다. 실제 고객은 객체의 인스턴스이다.(고객을 객체라고 부르기도 하지만) 그러므로 각 인스턴스가 동일한 속성의 집합을 갖게 되고 각 인스턴스는 이들 속성 각각에 대해 다른 값을 갖게 된다.(그림 2 참조)



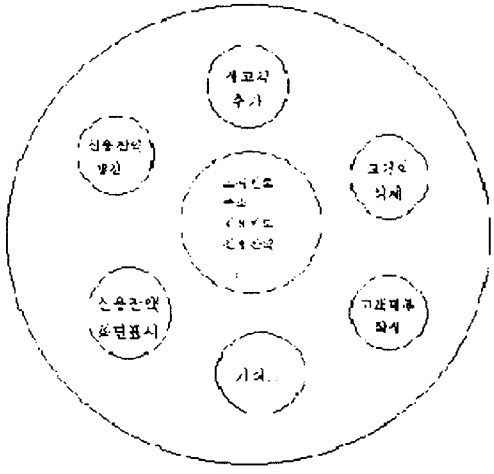
(그림 2) 고객 객체의 보기

객체의 자료를 조작하거나 보기 위해서는 객체의 메소드를 통하여야 한다. 그림 2에서 메소드는 새로운 고객의 입력, 고객의 삭제, 외상잔고나 고객명부의 갱신과 출력 등을 한다. 요컨대, 메소드는 허가 없는 접속으로부터 자료를 보호하고 이용자의 접속을 돕는다. 그림 3은 자료와 그 메소드가 함께 담겨진 꾸러미라고 볼 수 있는데 이것이 캡슐화의 원리이다.(그림 3참조)

각 객체는 적절한 방법을 가동하여 반응하는 일련의 메시지를 갖고 있다. 메시지는 객체가 계산을 수행하고 그 값을 갖도록 할뿐만 아니라 객체의 여러 속성에 할당된 값을 바꿈으로서 객체의 내용을 수정하기도 한다. 메시지는 객체나 다른 객체가 소유하는 메소드로부터 올 수 있다.

전술한 바와 같이 우리는 세상을 객체의 계층이라고 본다. 그림 4에서 보면 투자 포트폴리오는 부동산과 유가증권이라는 하위클래

속성과 메소드의 캡슐화



(그림 3) 속성과 메소드의 캡슐화

스를 갖는다. 부동산은 상업용과 주거용의 하위클래스를 가질 수 있다. 유가증권은 주식과 채권의 하위클래스를 가질 수 있다. 각 수준을 클래스라고 부른다.(Smalltalk 등에서는 객체 클래스라고 부른다.) 클래스는 객체를 정의하는 청사진과 같다.(Kurata 1997) 클래스는 하위클래스나 상위클래스를 가질 수 있다. 더구나, 각 클래스는 독자적인 속성과 메소드를 가질 수 있다.

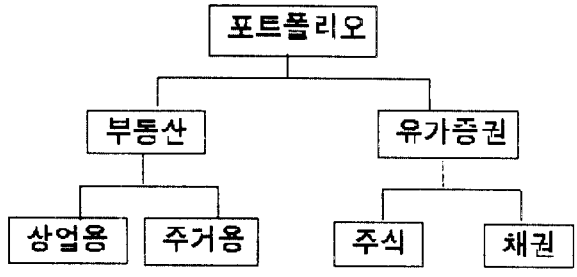
객체지향의 강력한 측면은 각 하위클래스가 상위클래스의 속성과 메소드를 상속할 수 있다는 것이다. 그림 4에서 부동산클래스가 적요, 소재지, 가격, 세금이라는 속성과 추가, 삭제라는 메소드를 갖고 있다면 상업용이나 주거용의 하위클래스에서 이를 재 정의할 필요가 없다는 것이다. 오히려 이 하위클래스를 정의하기 위하여 필요한 일은 하위클래스를 유일하게 하는 속성과 메소드를 추가하는 것이다. 사실, 다른 사물을 정의하므로서 새로운 하위 클래스를 정의할 수 있는 것이다.(그림 4 참조)

다른 객체클래스는 각기 동일한 이름의 메소드를 가질 수 있다. 이런 점을 다형성이라 한다. 예를 들면, 채권과 주식이라는 객체는 수익률이라는 메소드를 가질 수 있다. 채권의 경우 이자를 채권의 장부가격으로 나누어서 계산하고, 주식의 경우는 배당을 주식의 가격으로 나누어서 계산한다.

다형성은 메시지를 쓰기 쉽도록 해주는데 그 이유는 우리가 동일한 메시지를 각 객체에 보내서 실제로 어떻게 수익률을 계산하는지 고려하지 않고도 수익률을 계산하도록 한다.

문헌에는 아직까지 진정한 객체지향 분석과 설계를 위해 무엇이 필요한지와 객체지향 프로그래밍 언어를 구성하는 것이 무엇인지에 대해서는 완전한 합의가 없다.(Khoshafian 1991)

투자자산의 계층



3. OODM의 장단점

시스템 개발에 객체지향 접근법을 활용하는 장점이 많이 있다. 나중에 설명하겠지만 이들 각각은 시스템의 감사를 위하여 의미하는 바가 있다.

장점들은 다음과 같다.(IDC 1991)

① 시스템의 모형이 실상과 매우 닮았다. 이는 이용자와 개발자간의 의사소통을 용이하게 한다.

② 객체/메시지 패러다임은 복잡한 세상을 쉽게 모형화할 수 있다. 이것은 시스템이 보다 복잡하게 모형화되어 감에 따라 필수적이다.

③ 객체지향은 행동을 실체의 소유물로 봄으로 전통적인 E-R 모형의 틀을 넘고 있다. 더구나, 자료흐름도를 통하여 쉽게 구현할 수 있다.

④ 객체지향은 코드의 재사용을 지원한다. 독자적인 클래스 라이브러리(예를 들면, 계정도표, 구매주문서, 포트폴리오 등)를 구축한 회사도 더러 있다. 이들은 저장되어 나중에 수정하거나 재활용하게 된다. 일례로 어떤 조직에서 고객 객체 클래스를 만들었다고 하자. 고객이 필요로 하면 언제든지 이 클래스를 재사

용 할 수 있다. 구매요청서 객체 클래스가 만들어 졌다면 구매주문서 객체 클래스를 정의하기 위해서는 약간 수정하면 될 것이다.

⑤ 계층 라이브러리는 상업적으로도 가용하다. 일반 회계 객체(송장, 구매주문서 등)를 담고있는 클래스 라이브러리는 구매하여 직접 사용하거나 수정하여 쓸 수 있다.

⑥ 새로운 객체는 기존의 객체를 상속하여 만들 수 있다.

⑦ 재사용과 상속, 구입 등으로 시스템의 개발을 쉽고 용이하게 할 수 있다.

⑧ 프로그램과 시스템의 유지보수는 바꿀 필요가 있는 객체에만 집중함으로서 쉽다. 모든 객체는 메시지를 전달하여 상호작용하기 때문에 한 객체 클래스는 다른 객체 클래스의 수정으로 인하여 영향을 받지 않는다.

⑨ 객체지향은 적어도 이론적으로는 분산처리에 잘 맞는데 객체는 시스템 내의 어디에도 존재할 수 있기 때문이다.

객체지향자료모형의 단점은 캡슐화와 관련되어 있는데, 캡슐화는 객체코드를 새로운 응용프로그램에 재사용하게 하여 프로그래머의 효율성을 증대시킨다.

그러나, 한편으로는 극단의 경우 캡슐화의 원리는 수시 조회를 하는 능력을 배제시키는데 그것은 객체를 만들 때 시스템 설계자가 사전에 정의한 객체에 접속하고 객체를 조작하는 메소드를 제한하기 때문이다.

객체지향 DBMS에서 포인터를 사용하는 것도 문제가 된다. 이것은 이용자로하여금 명시적으로 정보를 검색하기 위하여 DB를 통하여 항해할 것을 요구하는 것이다. 관계형 자료모형의 지지자들은 이것이 엔드유저들의 입장이 아닌 프로그래머들의 편의를 도모하는 DB 구축방법으로 한 단계 퇴보한 것이라고 본다. 결국 객체지향 DBMS의 심각한 문제라면 SQL과 같은 쓰기 쉬운 조회언어, 표준의 부족이라고 본다.

제 3 장 OODM과 내부통제제도

내부통제제도의 수행은 내부통제의 비용을 유발시킨다. 이 비용이 내부통제로 인하여 오류와 부정을 예방하고 적발함으로서 절약되는 기회비용을 초과해서는 안된다. 또한 내부통제의 평가결과에 따라 감사의 정도(시간, 인력, 비용 등)가 결정된다.

내부통제는 조직전반에 걸친 사항과 보안에 관한 통제(General Control; GC)와 업무처리에 관한 통제(Application; AC)로 나눌 수 있는데 이를 EDP 감사의 기능별로 연계한다면 GC는 컴퓨터 환경감사(auditing around the computer)와 연계되고 AC는 업무처리과정 감사(auditing through the computer)와 연계된다. 본 연구에서 다룰 네 가지 EDP 감사기법은 컴퓨터를 활용한 감사(auditing by the computer)라고 할 수 있다.

이 논문의 목적이 기존시스템의 회계감사를 위해 개발된 EDP 감사기법을 객체지향 시스템을 감사하기 위해서는 어떻게 수정 보완해야 하는가를 연구하는 것이다. 당연히 감사의 목적은 변하지 않으며 재무제표가 제공하는 정보의 공정성을 검증하는 것이다. 따라서 열 가지 GAAS를 적용하는 것도 그대로이다. 감사는 내부통제제도를 검토, 평가하여 그 결과에 따라 필요한 만큼 회계정보를 검사하는 것이다.

1. 시스템의 개발과 내부통제

어떤 EDP 시스템과 마찬가지로 시스템 개발 과정에 대한 일반통제는 강화되어야 한다. 시스템을 문서화하는 데에 체계적인 방법이 사용되어야 하고 프로그램의 변경도 통제되어야 한다. 모든 프로그램은 충분한 검사를 받아 공식적으로 실행하기 전에 승인을 받아야 한다.

가. 시스템 설계의 질

모든 시스템은 설계한대로 만들어진다. 객체지향 모형은 우리가 세상을 보는 방법과 유사하므로 시스템의 이용자와 분석가가 의사소통을 잘해야 한다. 이렇게 해야 오해를 줄이고 최종 시스템에 오류가 생길 위험을 감소하게 되는 것이다. 또한 시스템의 설계를 감사인이 이해하고 평가하기 쉽도록 해야한다는 것을 뜻한다.

객체지향시스템에서 종래의 개념도표와

기본으로 할 수 있는가 하는 점도 검토해야 한다. Adamson과 Dilts(1995)는 회계시스템이 기본적인 회계사상을 직접 반영하도록 구축되어야 한다고 하였다. 여기서 의미하는 것은 감사인이 감사 하는 기본적인 회계시스템에 개념적 뷰를 달리 적용할 필요가 있다는 것이다.

나. 시스템 문서화

모든 시스템은 시스템과 프로그램의 흐름도, 자료흐름도 등을 포함하여 적절하게 문서화되어야 한다. 객체지향 시스템에도 마찬가지로 적용된다. 바뀌어야 할 것은 클래스 구조를 포함하여 객체와 객체들간에 교류되는 메시지를 표현하는데 쓰이는 기법이다. 객체지향 시스템과 프로그램의 문서화 포맷을 제시한 사람들은 Booch(1991) 등이다(Coad and Yourdon 1991).

다. 프로그램의 변경

객체지향 시스템이 모듈을 중심으로 한 것이기 때문에 프로그램의 변경도 쉽게 할 수 있어야 한다. 급여 프로그램에서 시간급과 월급을 받는 두 종류의 종업원이 있다고 하자. 모든 판매사원은 월급과 수당을 받기 때문에 판매사원 클래스는 봉급자의 하위클래스가 된다. 여기에는 수당 계산을 위해 필요한 메소드와 속성을 포함한다. 메소드는 클래스를 위하여 정의되기 때문에 수당 계산 메소드는 클래스의 모든 인스턴스에 쓸 수 있도록 투입되어야 한다. 회사가 판촉수당 계산방법을 바꾸려고 한다면 프로그래머는 판매원 클래스로 가서 수당 계산 방법을 수정하면 된다. 판매원이나 봉급자의 메소드는 영향을 받지 않는다. 판매원 객체에서 수당 정보를 요청하는 객체는 그 정보를 요구하는 메시지가 바뀌지 않기 때문에 수정될 필요가 없다. 급여의 모든 측면을 포함하는 큰 급여프로그램을 수정하려는 것이 이래서 쉽다는 것이다. 그러므로 감사인은 급여프로그램이 보수된 후에도 적절하게 기능을 발휘하고 있다는 확신을 갖게 된다.

라. 상속된 방법과 변수들

그러나 객체지향 시스템의 프로그램 변경측면에서 시스템의 위험문제를 증가시킬 수 있는 경우가 있다. 전술한 바와 같이 각 하위클래스는 각 상위클래스의 속성과 메소드를 상속한다. 이 경우에 내재하는 위험은 클래스 계층의 한 수준에서 새로운 메소드를 추가하거나 메소드를 수정하면 하위클래스에 적용될 때 예기치 않은 결과가 일어날 수 있다는 것이다. 이러한 위험을 경감시키는 방법은 모든 클래스 수준에서 변경된 프로그램 모두를 철저하게 검토하는 수밖에 없다. 예를 들면 급여 프로그램을 변경했을 때 결과가 예상대로 나오는지를 확인하기 위하여 모든 종업원의 클래스를 검사해야 한다.

Kandelin과 O'Leary(1995)가 밝힌 또 다른 문제는 일부(부분집합)를 복수의 상위 클래스에서 상속을 했을 경우 속성과 메소드간에 갈등이 생길 수 있다는 것이다. 이러한 위험을 감소시키는 방법은 시스템 설계를 신중히 하는 일이다.

라. 상업용이나 기존 클래스 라이브러리의 이용

경영과 회계를 위한 일반클래스 라이브러리를 구입하여 쓸 수 있다. 믿을만한 업자로부터 구입하였다면 내부통제 문제는 요소요소를 철저히 검토하여 약속대로 작동하게 해야한다. 이것은 회사내의 다른 부서에서 개발한 객체를 사용하는 경우도 같다. 개발 초기에 철저한 검사를 해야한다는 것이다.

상용이나 기존 클래스 라이브러리를 쓸 때 감사를 할 몇 가지 착안사항은 다음과 같다.

① 감사인이 이 객체들에 오류가 없다고 생각할 지라도 객체를 확장하려고 시도한 새 하위클래스는 검사할 필요가 있다.

② 객체가 설계한대로 작동할 지라도 현 상황에 완벽하게 맞지 않을 위험은 상존한다. 이것은 현 상황이 얼마나 독특한가 하는 문제이다. 조직내의 한 부서를 위해 개발된 급여 클래스와 객체는 다른 부서에서는 맞지 않을 수도 있다는 것이다. 하위클래스가 속성/메소드를 상속할 때 이 라이브러리가 예기치 못한 결과를 일으킬 가능성은 있다. 모든 경우에 위험을 줄이기 위한 방안은 철저한 검사를 하는 것이다.

2. 업무처리 과정의 통제

가. 자료입력의 통제

기존의 시스템에서는 자료를 입력하여 시스템의 일부가 되기 전에 자료에 존재 가능한 오류를 적발할 수 있도록 프로그램하여 범위검사, 합리성 검사, 타당성 검사 등을 실시하여야 한다.

예를 들면 신용판매시에는 고객번호를 읽어 타당한 번호인지를 알아내야 할 것이다. 당연히 오류가 있는 자료는 객체지향 시스템에서도 중요한 문제이다. 프로그램 된 편집체크의 개념이 바뀌지 않았다고 해도 수행되는 방법은 다르다. 수행방법에는 적어도 다음과 같은 세 가지가 있다.

① 계층 내에 통제클래스를 만든다. 이 클래스는 여러 가지 편집체크를 수행하는 방법을 갖고 있다. 계층에 있는 모든 하위클래스는 이 메소드들을 상속하게 된다. 메소드는 모든 입력자료에 사용될 수 있다.

② 영향을 받는 각 객체의 캡슐화된 메소드에 통제 메소드를 추가한다. 이것은 클래스 계층의 상위 층에서 이루어 져야 하는데 통제를 필요로 하는 하위클래스가 접속할 수 있도록 하려는 것이다. 고객이 신용으로 구매하거나 지불을 하거나 반품을 하거나 간에 고객의 계정번호가 타당한지를 검사해야 한다.

③ 개별 메소드에 통제코드를 추가한다. 범위검사, 한도검사, 타당한 코드검사 등 무결성의 검사는 메소드에 포함된다. 즉, 각 판매에 대한 합리성이 조사되어야 한다는 것이다.

나. 시스템 내부의 처리에 관한 통제

자료에 승인 없이 접근하지 못하도록 하는 것이 중요하다. 객체지향 시스템에서 접근은 객체에 메시지가 지나감으로서 일어난다. 즉 캡슐화는 사전에 정의되지 않은 메시지로 객체가 조작되는 것을 막는다. 이로 인하여 승인 없는 자료의 조작을 현저히 감소시키게 된다. 그러나 그것은 오류가 있는 메시지를 받을 수도 있기 때문에 객체가 메시지를 받는 시점에서 적절한 통제의 중요성을 강조하고 있다. 여기에 필요한 몇 가지 통제가 있다.

① 통제 클래스를 만든다. 이 클래스는 들어오는 메시지가 객체와 상호 작용하기 위하여 허가 받은 것인지를 감시하게 된다.

② 객체에 메소드를 추가한다. 이것은 들어오는 메시지가 자료와 상호 작용하기 위하여 방법상 승인된 것인지를 검증한다.

③ 개별 메소드에 코드를 추가한다. 특정 자료가 승인 받지 않은 조작과 많은 관련이 있는 경우 밝혀져야 한다. 이 상호작용을 요구하는 메시지가 타당한지를 검증하기 위하여 이 자료와 상호작용 하는 메소드에 코드를 추가해야한다.

④ 객체는 메시지를 여과한다. 객체는 메시지에 있는 요소들의 수가 완전한지를 검사하기 위하여 만들어져야 한다.

또한 감사인은 시스템이 메시지 DB에 유지해 온 모든 메시지의 사본이나 특정 요건을 충족하는 사본을 요구할 수도 있다. 그러고 나서 감사인은 감사절차의 일부로 이 DB를 검토하게 된다.

이상에서 살펴본 것은 객체지향이 내부통제의 개념을 변경시키지 않았다는 것이다. 그러나 실무상 특정 개념이 수행되는 방법에 영향을 줄 수는 있다. 언제나 감사인이 감사 계획을 수립할 때 내부통제에 설치될 신뢰성을 결정함에 있어서 통제와 적절성을 평가할 필요가 있다는 것이다.

제 4 장. OODM의 EDP 감사 기법

전통적 시스템에서 자료와 프로그램을 감사하는데는 여러 가지 기법이 있다.

EDP 감사는 감사객체인 시스템과 자료가 고객의 실제 시스템과 자료인지 감사인이 준비한 시스템과 자료인지에 따라 달라진다. 다음 표 1과 같이 고객의 프로그램으로 감사인의 데이터를 활용하는 기법이 입출력을 중시하는 방법이고, 감사인이 프로그램이나 감사모듈을 준비하여 고객의 실제자료로 감사하는 처리 과정을 중시하는 방법이 있다.

테스트 데이터법(Test Data; TD), ITD/ITF(Integrated Test Data/ Integrated Test Facility; ITD/ITF)법은 입출력을 중시하는 방법이고 EAM(Embeded Audit Module)과 GAS(Generalized Audit Module)는 처리

(표 1) EDPA의 구분

방법 \ 소유		데이터		프로그램	
		의뢰인(고객)	감사인	의뢰인(고객)	감사인
입출력 중심 감사	TD	-	o	o	-
	ITD/ITF	o	o	o	o
처리 중심의 감사	EAM	o	-	o	o
	GAS	o	-	-	o

과정을 중시하는 방법이다.

이 장에서는 주요한 네 가지 감사기법을 객체지향 시스템에 적용할 때 야기될 수 있는 문제를 연구하려고 한다.

1. 입출력중심의 감사 방법

가. 테스트 데이터법

TD 접근법에서는 감사인이 타당한 거래와 타당하지 않은 거래 모두를 포함하는 미리 준비한 시험자료를 처리하기 위해 고객의 S/W를 이용한다. 처리결과는 감사인이 예기한 것과 비교한다. 타당한 거래와 타당하지 않은 거래(누락자료, 틀린 종업원 번호 등)를 포함하는 것은 시스템의 프로그램 된 업무처리 통제를 검사하는 것이다. TD 방법은 컴퓨터를 암실로 본다. 감사인이 처리할 거래를 알고 결과가 예상한 대로 확인되었다면 고객의 업무처리가 옳다고 보는 것이다. TD가 감사인에게 중요한 관련사항인 오류와 예외사항을 적발토록 충분히 널리 적용되는 한 이 기법은 기존의 시스템이나 객체지향 시스템 모두에 같은 방법으로 적용될 것이다. 그러나 객체지향 시스템에서는 상속 때문에 감사인이 시스템에 존재하는 각 하위 클래스에 타당한 거래와 타당하지 않은 거래를 포함시키는 것에 주의해야 한다.

통제를 검사하기 위하여 시험자료를 이용하는 것은 통제절차가 프로그램에 깊이 내장되어 있거나, 처리량이 많아서 시스템의 성능(과)이나 통제를 검증하는 것을 전적으로 수작업에 의존해야 하는 경우에 유리하다.

시험자료를 적절하게 개발하기 위해서는 검사할 과정을 철저하게 파악하여 고객의 시스템과 프로그램이 오류와 부정을 예방하고 적발할 수 있도록 설계되었다는 것과 시험자료가 처리조건, 거래의 형태, 레코드 등을 대표할 수 있도록 해야 한다.

시스템을 개발할 때에 시험과정을 문서화하여두면 감사인에게 시험자료로 쓸 원천을 잘 제공할 수 있게 된다. 사용할 시험자료의 사용시기와 통제 방법, 사용하는 프로그램의 버전, 테스트를 할 때에 운영환경을 적절히 통제하는 것 등이 중요한 고려사항이다.

나. ITD/ITF

TD의 문제는 감사인이 검사한 S/W를 고객이 실제로 사용했는지를 보증할 수 없다는 것이다. ITF에서는 감사인이 TD와 고객의 실제 자료를 관리자가 모르는 채 통합하여 예상한 것을 출력과 비교한다. 1년에 수차례 ITD를 처리함으로서 감사인은 고객이 감사 받는 S/W를 실제로 사용하고 있음을 보증한다. ITD 기법에서의 문제는 ITF의 사용이다. ITF는 고객의 시스템 내에 만들어진 가공의 실체(가공의 고객, 업자, 재고 항목, 제품 등)이다. 다시 감사인은 처리결과를 예상치와 비교한다. ITD의 이용과 마찬가지로 감사인은 고객이 감사 받은 S/W를 실제로 사용하고 또 제대로 기능을 발휘하고 있다는 것을 보증하게 된다. 앞서 논의한 TD와 마찬가지로 ITD와 ITF는 컴퓨터를 암실로 간주한다. 감사의 관점에서 보면 이 기법들은 기존 시스템이나 객체지향 시스템에 같은 방법으로 적용될 것이다. 그러나 ITF의 구현은 흥미롭다. 가상의 고객을 만들기 위해서는 그 고객 객체에 새로 인스턴스를 만들어야한다. 그러나 감사인의 행위가 노출되지 않는 가상의 감사 클래스를 만드는 것이 성패의 관건이다.

ITF 프로그램의 목적은 원시자료 통제와 처리통제를 검사하는 것이다. ITF를 통과할

(표 2) TD법의 장단점

TD법의 장점	TD법의 단점
사용하기에 폭넓은 프로그램지식을 요구하지 않는다.	모든 오류가능성을 검사할 수는 없다.
내부감사인이 쉽게 이해한다.	복잡한 시스템에서 입력자료와 출력보고서를 연계시킬 수 없다.(특정 시스템의 출력은 저장매체에 메모리 되어 있다.)
전시스템을 검사할 수 있다.	
결과를 쉽게 확인할 수 있다.	
자료를 처리함에 있어 시스템의 정확성에 관한 여론이 형성될 수 있다.	시스템에 존재하는 견제와 내부통제에 관한 의견이 형성되지 않는다.
정규컴퓨터 프로그램을 사용할 수 있다.	독립된 file을 쓰지 않는다면 시스템에서 시험자료를 역류시키거나 환원할 수 없다.
컴퓨터 주변환경을 감사할 때 존재하지 않는 상황을 검사할 수 있다.	만족할 만한 시험거래를 준비하는 데에 시간이 오래 걸린다.
감사인은 경험을 쌓을 수 있다.	시험거래를 준비하는데 기술적 지식이 필요하다.
감사인은 검사에 대한 통제를 유지할 수 있다.	
컴퓨터 사용 시간을 줄일 수 있다.	

시험거래를 준비하는데 시험거래는 더미장치가 에뮬레이트하는 모든 거래를 대표해야 한다. 정상적인 조건하에서 오랜기간동안 시스템을 적절하게 검사하기 위해서는 타당한 거래와 타당하지 않은 거래 모두가 정규거래와 섞여 사용되어야 한다.

실제결과와 예정된 결과를 대조할 수 있도록 모든 출력과 처리루틴을 심사한다.

2. 처리중심의 감사 방법

가. EAM

고객이 대량의 거래를 처리하는 경우 감사인은 합당한 검사와 분석을 하기 위하여 고객이 준비한 자료를 선별하려는 목적으로 고객의 업무처리 S/W에 감사모듈을 설치하는 수가 있다. 예를 들면 모듈은 고객 계정의 거래를 감시하여 감사인이 지정한 일정액 이상의 거래를 계속적으로 선택할 수 있다. 이렇게 선별된 거래는 나중에 분석하기 위하여 따로 저장된다. 객체지향 시스템에서 EAM을 이용하는 목적은 바뀌지 않을 지라도 기법이 구현되는 방법은 바뀌게 된다. 객체지향 시스템에 EAM을 구현하는 방법은 몇 가지가 있다.

① 클래스계층에 감사모듈 클래스를 만든다. 모든 하위클래스는 자동적으로 상위클래스의 방법을 상속하기 때문에 이 모듈은 다양한 형태의 거래를 감시할 수 있다. 이 방법에는 두 가지 문제가 있다. 첫째, 감사모듈을 클래스 계층내의 분리된 클래스로 만듦으로서 감사인은 감사도구에 주의를 해야한다. 둘째, 누가 하위클래스에 감사모듈의 메소드를 무력하게 할 같은 이름을 가진 메소드를 삽입할 위험이 상존 한다는 것이다.

② 객체내부에 캡슐화할 감사방법을 만든다. 예를 들면, 모든 들어오는 메시지를 감시할 메소드를 판매 객체에 포함시킬 수 있다는 것이다.(거래 메시지는 객체에 보내진다.) 예를 들어 거래가 백만원 이상이라고 하면 복사하여 감사 객체로 분리시킨다. 감사인은 적절한 메시지를 메소드에 보냄으로서 메시지를 작동케 하거나 중지토록 할 수 있다.

③ 객체의 기존 메소드를 수정한다. 예를 들면, 백만원 이상의 거래에 대한 메시지를 받으면 기존의 판매 객체 메소드에 있는 코드는 자동적으로 감사객체에 사본을 보낸다. 이것은 분리된 감사 메소드를 만드는 것보다 더 복잡하다.

나. GAS

GAS 패키지는 전통적으로 고객의 자료파일에서 감사절차를 수행할 루틴의 집합을 제공한다. 예를 들면 GAS는 레코드가 서류에 기록되어 있을 때 감사인이 할 일을 컴퓨터 파일에 저장되어 있기 때문에 대신한다고 보면 된다. GAS는 쓰기 쉬운데 감사인이 GAS를 쓰기전에 접속할 고객의 파일, 수행할 업무, 원하는 출력형식 등을 구체화하는 S/W 명세표를 만들어야 한다.

객체지향 환경에서 GAS의 개념은 여러 가지 감사기능을 수행할 메소드를 포함하는 감사 객체 클래스를 만듦으로서 구현된다. 예를 들면 감사 객체는 재고잔액을 분기 위하여 재고 객체에 메시지를 보내거나, 재고 객체에

개별항목의 잔액을 묻고, 감사 객체에 이를 저장하고, 합계를 계산할 수 있다. 다른 메소드는 재고합계를 재무제표 객체의 재고합계와 비교할 수도 있다.(Richhiute 1992)

기존의 환경에서 GAS를 쓰는 것과 객체지향 환경에서 감사 객체 클래스를 이용하는 데는 현저한 차이가 있다. 전통적으로 GAS는 고객의 자료파일에 직접 접속한다. 그러나 캡슐화 때문에 감사 객체는 그 목적을 이루기 위해 메시지를 보내어 다른 객체들과 작용하게 된다. 객체에 있는 메소드는 실제 자료 검색을 수행한다. 그러므로 감사 객체와 고객의 자료간에는 고객 S/W(메소드)의 층이 있다. 감사인이 그의 고객 객체에 있는 메소드에 의존하려 하지 않는다면 이것이 객체지향 환경에서 GAS를 사용하는 한계가 될 것이다.

그러나 GAS의 장점은 다음과 같은 점이다. 반복적인 일에 소비하는 노동시간을 줄여서 감사를 효율화한다. 즉 합계계산, 보고서간의 자료복사, 보고서 작성 등이 쉽게 처리된다. 표본추출이 임의적임으로 편의(偏倚)를 배제하여 임의성을 향상시키고 이로 인하여 표본의 정확성, 신뢰성, 감사의 정확성에 직접 영향을 준다. 컴퓨터가 만든 보고서와 일정계획은 보다 객관적이며 전문적이다. 감사인은 보다 많은 자료를 검토할 수 있고 원하는 자료를 보다 많이 추출할 수 있다.

3. 절충법

앞에서 GAS가 필요한 자료를 감사 객체로 보내기 위해 고객의 S/W 메소드에 의지해야 한다는 한계를 갖고 있다고 하였다. 객체지향 환경에서의 감사는 전통적 시스템을 감사하는데 쓰이는 EAM과 GAS를 절충한 새로운 접근법이 필요하다. 이 절충법으로 감사 객체 클래스를 만들어야 할 것이다. 이들 메소드는 (그렇게 하려는) 메시지가 감사 객체 클래스에서 접수되면 객체의 자료를 검색하거나 조작할 것이다. 메소드에 감사 메시지를 보내는 감사 객체 때문에 순효과는 객체지향 환경 하에서 감사인이 직접 접속을 할 수 있다는 것이다.

제 5 장 결론

객체지향 모형은 요즈음 시스템이 복잡하고 객체지향 접근법의 장점이 많기 때문에 시스템 개발에 강력한 힘을 갖고 있다. 이 논문은 전통적 EDP 감사도구와 기법을 새로운 환경에 적용하려는 연구이다. 감사의 목적과 GAAS는 여전히 그대로 적용된다. 전반적인 내부통제의 목적과 감사인이 내부통제를 평가, 검토하는 필요성도 그대로 바뀌지 않고 있다. 그러나 객체지향 환경에서는 그 성질상 객체지향 시스템이 다른 위험을 감소시킬지라도 전통적 시스템에서는 없던 위험이 존재할 가능성이 있으므로 이에 대한 연구가 필요하다.

TD, ITF 등과 같은 시스템의 입출력에 초점을 두는 전통적 EDP 감사기법들은 아주 적용하기가 쉽다. 그러나 EAM이나 GAS 등 처리에 초점을 두는 기법들은 구현을 하기 위해 다른 메소드를 필요로 한다. 감사인에게 강력한 도구를 제공하기 위하여 EAM과 GAS의 개념을 결합하는 새로운 절충형 접근방법이 필요하다고 본다. GAS가 EAM의 메소드를 자유로이 활용하게 되면 객체의 자료조작과 검색의 효율이 증가하게 되는 것이다.

REFERENCES

AICPA, 1991. *Codification of Statements on Auditing Standards*. New York: AICPA (AU Sec. 150)

AICPA, 1994. *Auditing With Computers*.

Adamson, I. L. and D. M. Dilts 1995. Development of an Accounting Object Model for Accounting Transactions. *Journal of Information Systems*(Spring): 43-64.

Ambler, Scott W. 1995. *The Object Primer*. New York: SIGS Books.

Andleigh, P K. and M. R. Gretzinger, 1992. *Distributed Object-Oriented Data-Systems Design*. Englewood Cliffs, NJ: Prentice Hall.

Bell, T. B., F. O. Marrs, I. Solomon, and H. Thomas. 1997. *Auditing Organizations through a Strategic-Systems Lens*. KPMG Peat Marwick LLP.

Booch, G. 1991. *Object-Oriented Design with Applications*. Redwood City, CA: Benjamin/Cummings Publishing Company, Inc.

Coad, P., and E. Yourdon. 1991. *Object-Oriented Analysis*, 2nd ed. Englewood Cliffs, NJ Yourdon Press.

Courtney, J. F., and D. B. Paradice. 1992. *Database Systems for Management, 2nd ed.* Homewood, IL: Irwin.

Duntemann, J. 1990. Our Object All Sublime. *PC Techniques* (April/May): 14-22.

Elmasri, R., and S. B. Navathe. 1989. *Fundamentals of Database Systems*. Redwood City CA: The Benjamin/Cummings.

English, L. P. 1992. Object Databases at Work. *DBMS* (Oct): 44-58.

IDC White Paper. 1991. Object Technology: A Key Software Technology for the '90s. *Supplement to Computerworld*.

Kandelin, N. A. and T. W. Lin. 1992. A Computational Model of an Events-Based Object Oriented Accounting Information System for Inventory Management. *Journal of Information Systems* (Spring): 47-62.

Kandelin, N. A. and D. E. O'Leary. 1995. Verification of Object-Oriented Systems: Domain-Dependent and Domain-Independent Approaches. *Journal of systems and software* (June): 261-269.

Khoshafian, S., 1991. Modeling with Object-Oriented Databases. *AI Expert* (Oct): 27-33.

Khoshafian, S., and R. Abnous. 1990. *Object Orientation: Concepts, Languages, Databases, User Interfaces*. New York, NY: John Wiley.

Kurata, D. 1997. Learn the Fundamentals of OOP. *Visual Basic Programmer's Journal*. (June): 81-84.

Marshall B. R., P. J. Steinbart, and B. E. Cushing. 1997. *Accounting Information System*. Addison-Wesley.153-157.

Martin, J., and J. Odell. 1992. *Object-Oriented Analysis & Design*. Englewood Cliffs, NJ: Prentice Hall.

McCarthy, W. E. 1982. The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment. *The Accounting Review* (July): 554-577.

Murthy, U. S. and C. E. Wiggins, Jr. 1993. Object-Oriented Modeling Approaches for Designing Accounting Information Systems. *Journal of Information Systems* (Fall): 97-111.

Rasmus, D. W. 1992. Relating to Objects. *BYTE* (Dec): 161-165.

Richhiute D. N. 1992. *Auditing*, 3rd ed. Cincinnati, OH: South-Western.

Smith, D. N. 1991. *Concepts of Object-Oriented Programming*. New York, NY: McGraw-Hill.

Taylor D. A. 1990. *Object-Oriented Technology: A Manager's Guide. Reading*, MA:Addison-Wesley.