# 전사적 테스트 주도 개발방식의 적용 경험과 교훈

**Oh-Seung Kwon[a] and Joa-Sang Lim[b] and Sa-Neung Hong[c]**

*[a] 서울시립대학교, 대학원, 경영학과*
*130-743 서울특별시 동대문구 시립대길 13(전농동 90번지)*
*Tel: +82-2-2210-2114, Fax: +82-2-2210-2114, E-mail: kos@componentbasis.com*

*[b] 상명대학교, 소프트웨어대학, 디지털미디어학부*
*110-743 서울특별시 종로구 홍지동 7*
*Tel: +82-2-2287-5114, Fax: +82-2-* 2287-0023, *E-mail: jslim@smu.ac.kr*

*[c] 서울시립대, 경상대학, 경영학부*
*130-743 서울특별시 동대문구 시립대길 13(전농동 90번지)*
*Tel: +82-2-2210-2114, Fax: +82-2-* 2210-2114, *E-mail: snhong@uos.ac.kr*

## Abstract

선행적이며 지속적인 테스트를 강조하는 테스트 주도개발이 시스템의 품질과 개발 생산성을 제고한다는 연구가 보고되고 있다 [4]. 그러나 대부분의 연구는 실험실 환경 또는 소규모 프로젝트를 대상으로 수행하였고, 개발 현장에서의 효과를 검증한 결과는 드물다 [2, 5, 18].

본 연구는 비즈니스 환경의 대규모 프로젝트에 테스트 주도의 개발방법을 적용한 경험과 교훈을 보고한다. 다양한 데이터, 업무간 복잡한 연계, 철저한 검증의 필요성과 같은 전사적 응용체계의 요구사항은 기존의 테스트 주도 개발방법을 그대로 적용하기 어렵게 한다. 따라서 본 연구에서는 테스트 주도 개발방법의 전사적 적용을 위한 프레임워크를 제안하고, 이를 기반으로 한 테스트 지원도구를 개발하였다. 도구는 GUI 기반의 테스트 관리 화면을 제공하고 관계형 데이터베이스에 테스트 데이터를 저장하여 테스트 케이스의 생성, 테스트 실행, 그리고 테스트 데이터의 관리를 지원하였다. 도구는 또한 스크립트 방식이 아닌 저장된 테스트 데이터를 이용한 회귀 테스트의 실행을 가능케 하였다.

지원도구를 이용한 전사적 테스트 주도 개발은 테스트 결과의 실시간 파악과 빈번한 변경관리를 용이하게 하는 것으로 평가되었다. 또한 전사적 테스트 주도 개발방법의 보편적인 적용을 위해서는 전통적 개발방식에 익숙한 개발자들의 새로운 접근방법에 대한 거부감 해소, 테스트 주도 개발을 고려한 개발체계와 프로젝트 관리, 그리고 개발자 행태와 프로젝트 특성을 감안한 지원도구에 대한 후속 연구의 필요성이 식별되었다.

**Keywords:** 테스트주도개발 (TDD), 회귀시험, 테스트 지원도구, 전사적 테스트주도개발 (eTDD)

## 1. 서론

소프트웨어 개발에 있어 선행적이며 지속적인 테스트의 중요성이 강조되고 있다. 설계와 구현에 앞서 요구사항을 검증할 수 있는 테스트 케이스와 데이터를 선행적으로 작성함으로써 시스템의 품질을 조기에 확보하고, 나아가 비생산적인 재작업을 원천적으로 제거할 뿐만 아니라, 지속적으로 회귀 테스트를 수행하여 빈번한 변경관리를 용이하게 할 것으로 기대하기 때문이다. 최근의 연구도 테스트 주도의 개발 방법(Test-Driven Development)이 실제로 시스템의 품질과 개발 생산성을 제고한다고 보고하고 있다 [4].

그러나 아직도 대부분의 프로젝트에서는 설계와 구현을 종료한 이후에 테스트를 수행하는 전통적인 접근방법을 고수하는 것이 일반적이다. 이에 따라, 프로젝트 후반의 짧은 기간에 이루어지는 대규모 통합에 따르는 어려움을 겪어야 하는 것은 물론이거니와, 일시에 발견되는 많은 오류와 결함의 원인을 발견하고 수정하는데 감당하기 어려울 정도로 많은 자원의 소모를 반복하고 있다. 또한 일정이 급박한 상황에서 변경관리의 실패는 프로젝트의 품질 저하와 기한 불이행의 중대한 요인으로 지적되기도 한다. 비즈니스 환경의 대규모 프로젝트에 테스트 주도의 개발방법을 적용하기 위해선 아직 많은 연구와 지원이 필요함을 대변하는 현상이다.

본 연구는 국내 두 개의 대형은행을 합병하는 과정을 마무리 짓는 IT 통합 프로젝트에 테스트 위주의 개발방식을 적용한 경험과 교훈을 보고한다. 지금까지 테스트 위주 개발방식에 대한 연구와 지원이 대부분 실험실 환경이나 중소규모 프로젝트를 대상으로 하였다는[2, 5, 18] 한계점을 극복하기 위해서 이러한 연구는 반드시 필요하다.

연구는 저자가 프로젝트에 참여하여 직접 경험하고 관찰한 내용과, 프로젝트에 관련된 책자와 대외 발표자료 그리고 일부 내부 문건을 바탕으로 한다. 저자 중의 한 명은 상시 자문으로 전체 기간 동안에, 한 명은 지원도구 개발과 테스트 지원 인력으로 1년 반에 걸쳐 프로젝트에 참여하였다.

다음 2장에서는 테스트 위주 개발방법과 테스트 위주 개발을 지원하는 도구에 대한 기존의 연구를 살펴보고, 3장에서는 적용된 프로젝트에 대한 간략한 기술과 프로젝트의 대상인 은행 업무의 특성을 살펴본 뒤, 전사적 테스트 위주 개발방법 적용을 위한 프레임워크를 제안한다. 전사적 적용을 위한 지원도구는 4장에서, 그리고 전사적 적용 경험과 시사점은 5장에서 기술하고, 6장은 결론과 향후 연구과제로 마감한다.

## 2. 관련연구

지금까지 보통 테스트는 프로그램이 완성되면서 이를 검사하는 개발단계 이후의 사후적 방법으로 활용되어 왔다. TDD에서는 테스트를 개발단계 이전으로 앞당겨 설계를 구체화하는 사전적 도구로 발전시키고 있다. 즉 TDD에서는 테스트케이스와 스크립트를 우선 작성하고 프로그램을 개발한다. 그 후 테스트케이스를 실행하고 프로그램에 결함이 있는 경우에는 코드를 추가하게 된다. 이러한 방식은 테스트주도개발로 XP에서 제안되었는데 [1], 기존 방법론 (예: RUP)이 딱딱하고 상세설계를 강조한 것과는 달리 프로그래밍을 강조하고 있다. 이러한 방식의 TDD는 최근에 등장한 개념이 아니고 1960년 초반으로 거슬러 올라가 NASA 머큐리프로젝트에서 시도된 바 있다 [12]. 그러나 최근에서야 이에 관한 연구와 적용이 활발해지고 있다 [9]. 아래에 TDD 적용에 관한 연구를 정리한 표를 보면 대부분의 연구가 학생을 대상으로 이루어져 오고 있음을 알 수 있다. 실무에서 이루어진 연구는 불과 4편이 있다 [2, 5, 14, 20] 그러나 테스트가 중요시되는 실시간 업무처리 성격을 갖는 은행에 대한 TDD의 적용사례는 보고된 바 없다. 또한 TDD관련 연구의 결과가 일관적이지 못해 혼란이 있다. TDD가 생산성에 미치는 효과는 긍정적 [4, 10]이거나 부정적 [2, 5, 20]인 결과가 혼재되어 있고 몇 연구에서는 차이가 없었다 [14, 22, 23]. 반면 품질에 대한 영향은 보다 긍정적이어서 TDD가 품질을 저하시킨다는 부정적 결과는 없었다. 품질을 향상시키거나 [2, 5, 14, 21] 차이가 없는 것으로 나타났다 [4, 22, 23].

앞서 논의한 바와 같이 지금까지의 TDD 관련 연구는 소규모의 프로젝트에 국한되어 왔다. 반면 대규모 프로젝트에서는 엄격한 테스트를 위해 그 절차와 관련 산출물의 보다 체계적인 지원과 관리가 필요하다. 재개발 전에는 은행어플리케이션의 심각성에도 불구하고 테스트가 개발자 개인에게 의존되어 테스트케이스가 작성, 실행되고 결과기 보고되어 느슨하게 관리되는 임시적 (Ad-hoc) 경향이 있었다. 이는 최근 [6]가 캐나다의 테스트현황조사에서 나타난 바와 같이 테스트가 공학적 방법보다는 직관과 경험에 의존하고 있다는 결과와 많이 차이 나지 않고 있다. 기존 연구에서는 테스트의 복잡성을 감안하여 사람에게 의존하기 보다는 자동화와 관리시스템의 필요성을 제기하고 있다 [7, 11]. 또한 테스트활동의 지원도구는 1990년대 초반부터 그 사례가 보고되고 있다. 1992년 [13]는 UNIX에서 형상관리 RCS와 운용되는 테스트관리도구를 구현하였다. [3] 는 미국 통신회사인 BellSouth에서 개발한 TCMS를 통해 테스트케이스의 저장, 실행의 자동화와 진척의 조회기능을 구현하였다. [16] Object-Z 명세에서 테스트케이스를 추출하고 템플릿방식의 프레임워크를 제안하고 관리하는 시스템을 구현하였다. 본 연구와 같이 결함이 심각한 항공업무에서도 테스트관리도구의 필요성이 제기되어 구현사례가 보고된 바 있다 [15]. [19]에서는 기능테스트와 더불어 구조 (로직)의 테스트도 필요하다고 강조하였다.

표 1 : TDD의 품질-생산성 영향에 관한 연구

| 연구 | 대상 | 품질 효과 | 생산성 효과 |
|---|---|---|---|
| [2] | 산업체 | 긍정적 (약 2배) | 부정적 (15%) |
| [4] | 학생 | 차이 없음 | 긍정적 |
| [5] | 산업체 | 긍정적 (18%) | 부정적 (16%) |
| [10] | 학생 | 긍정적 | 긍정적 |
| [14] | 산업체 | 긍정적 (40-50%) | 차이 없음 |
| [20] | 산업체 | 결과 없음 | 부정적 |
| [21] | 학생 | 긍정적 (45%) | 결과 없음 |
| [22] | 학생 | 차이 없음 | 차이 없음 |
| [23] | 학생 | 차이 없음 | 차이 없음 |

## 3. 전사적 테스트 위주 개발 프레임워크

### 3.1 Big Bang

사례는 국내 두 개의 대형은행을 합병하는 과정을 마무리 지은 IT 통합 프로젝트이다. 참여인원 최대 1,400여명, 예산 3000억여 원에 이르는 초대형 프로젝트로 2003년 11월부터 2006년 11월까지 3년에 걸쳐 수행되었다. 일반적인 합병의 경우 주도권을 가진 조직에서 프로세스와 IT를 장악하여 합병되는 조직의 데이터를 전환하는데 비하여 연구의 배경이 되는 합병은 새로운 IT를 전면적으로 구축하여 조직 변환(transformation)에 성공한 사례이다. 또한 미국과 유럽의 금융 업계에서 불가능하다고 여기던 대형

은행 업무 전체를 개방형 서버로 구축하여 빅뱅 방식으로 전환하는 프로젝트를 훌륭하게 성공시킨 사례이기도 하다. 본 연구는 전체 프로젝트 중에서 중요도와 위험이 가장 큰 코어시스템의 구축에 테스트 주도 개발방식을 적용한 경험을 바탕으로 한다. (이하에는 연구 대상 시스템을 코어시스템으로 지칭한다.)

코어시스템은 수신, 여신, 외국환, 고객, 세무, 대행, 재무회계, 금융IC, 공통서비스, 자동이체 등을 대상으로 한다. 이는 은행이 고객과의 거래를 처리하는 시스템이다. 20백만의 고객에 대한 정보를 관리하고, 일 평균 20백만 여건, 초당 최대 3,000여건의 거래를 처리해야 하며, 연중 24시간 장애가 없어야 하는 등의 높은 성능과 고도의 신뢰성이 요구되는 시스템이다. 또한 다양한 은행 상품과 채널을 지원하기 위하여 설계, 구현한 10,000여 본의 총 10백만 라인에 이르는 응용 프로그램으로 구성된 매우 복잡한 시스템이다. 뿐만 아니라 코어시스템은 전사적 데이터웨어하스를 비롯한 60여 종에 이르는 내부 시스템은 물론 금융결제원을 비롯한 수백여 개의 외부기관과 지속적인 연계를 필요로 한다.

코어시스템의 설계와 구현은 정보공학 방법론을 기반으로 하였으나 대상 시스템의 중요성과 위험을 고려하여 구현과정을 3개의 타임박스로 분할하는 반복적 방법론을 혼용하였다. 또한, 품질의 조기 확보가 프로젝트의 성패를 결정하는 가장 중요한 요소 중의 하나임을 인식하여 테스트 주도의 개발방식을 적용하기로 하였다.

### 3.2 대규모 응용체계의 특징

테스트 주도 개발은 상세설계, 개발과 테스트를 병행하는 개발 프로세스이다. XUnit으로 대표되는 최근의 테스트 주도 개발은 기본적으로 함수 단위의 테스트와 코드 개발을 대상으로 하고 있다. 이러한 접근방법은 통합과 시스템 테스트에서는 유효성이 떨어지고, 은행 업무와 같이 개발 후반까지 많은 변화가 발생하는 경우에는 테스트 케이스의 유지보수가 불가능에 가까울 정도로 어렵게 된다 [17]. 뿐만 아니라, 사례와 같은 비즈니스 환경의 대규모 프로젝트는 비즈니스, 개발 프로그램, 시스템 환경과 테스트 환경 측면에서 단위함수 중심의 테스트 주도 개발방법을 적용하기 어렵게 하는 특징이 있다.

비즈니스 측면에서 볼 때, 은행은 국내 1,006개 지점, 국외 21개 점포로 구성되어 있고, 자동화기기는 CD기 1,284대와 ATM기 5,816대가 전내에 산재해 있다. 은행 어플리케이션은 DB 기반으로 개발되어 단순히 테스트 결과값만 비교하는 것이 아니라 테스트 실행 후 등록, 수정인 경우 테이블의 내용도 비교해야 한다. 이것은 단순히 테이블을 조회해서 Hard Copy를 작성하던 작업을 대체한다. 은행 어플리케이션은 함수의 규모가 10KLOC 이상으로 상당히 크며, 하나의 함수가 어플리케이션 자체가 될 정도이다. 은행 어플리케이션은 함수의 호출관계를 보면 호출 깊이가 깊은 계층구조이며 순환적인 특성이 있다. 또한, 해당 함수의 테스트를 위해서는 다른 여러 함수의 테스트가 선행되어야 할 필요가 있다. 은행 어플리케이션에서 함수 반환 값의 특성은 단순 변수에 의한 반환이 아니라 대부분 구조체(또는 객체)로 구성되어 있다. 은행 어플리케이션에서 함수는 복잡한 비즈니스의 요구를 수용하기 위해 입· 출력 파라미터의 개수가 수십 개에서 수백 개로 대단히 많아 일일이 입력하기 어렵다. 은행 어플리케이션은 미들웨어 또는 어플리케이션 프레임웍 기반에서 구축된다. 또한, 다양한 채널에서 들어오는 서로 다른 형태의 전문들은 채널 통합 단에서 받아 표준전문 형태로 변환하여 은행 어플리케이션에게 전송한다. 다양한 요구사항을 수용할 수 있도록 만들어진 은행 어플리케이션은 구조가 복잡하여 테스트 환경의 구축이 단순하지 않다. 관리적인 면에서 은행의 차세대 시스템 개발 시에는 전사적 규모로 수행되어 테스트 진행과정이 실시간으로 조회될 필요성이 있다. 반복적 개발을 먼저 하면서 장기간 개발에 따른 요구사항의 빈번한 변경으로 인해 완료된 시스템에 대해서 반복적으로 발생하는 테스트 투입공수를 절감하기 위해 테스트 자동화를 이용하여 전사적 테스트 주도 개발이(eTDD, Enterprise Test Driven Development) 필요하다.

### 3.3 eTDD 프레임워크

앞서 논의한 대규모응용체계의 특징으로 인해 은행어플리케이션에는 XUnit과 같은 도구를 사용한 TDD의 적용이 어렵다. 은행 어플리케이션에 대한 테스트 주도 개발을 위해 전사적 테스트 주도 개발(또는 eTDD)에 적합한 프레임웍을 그림 1과 같이 제시한다.

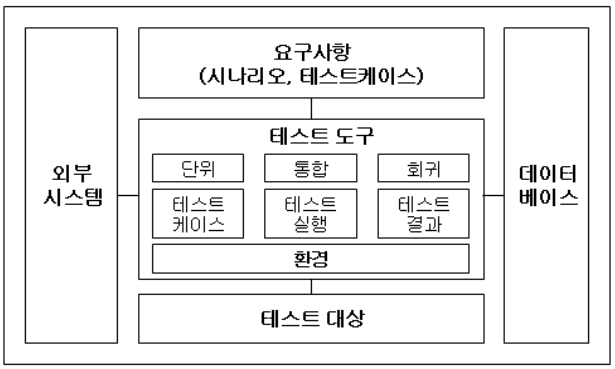


**그림 1 – eTDD 프레임웍**

위의 그림에서 제시한 프레임웍은 요구사항, 테스트 도구, SUT, 데이터베이스, 외부시스템의 컴포넌트로 구성하였다. eTDD에서는 테스트 베이스라인을 설정하기 위해 통합테스트시나리오와 단위 테스트케이스를 요구사항과 연계가 필요하다.

테스트 도구는 데이터 기반으로 복잡한 데이터로 구성된 테스트케이스를 작성할 수 있는 수단이 필요하다. 또한, 단위 테스트 요구사항을 관리하고 통합 테스트 요구사항을 관리 할 수 있는 수단이 필요하다. 단위 테스트케이스에서 작성된 케이스를 이용하여 통합 테스트 시나리오의 데이터를 구성하고, 회귀 분석 결과를 이용하여 이미 작성된 테스트케이스를 재사용하여 회귀테스트를 수행할 수 있는 수단이 필요하다. 테스트 도구에서 테스트케이스에 대한 생성 및 입력이 필요하며, 이에 관한 연구가 많이 있지만 은행의 경우에는 적합하지 않았다. 비즈니스가 복잡하므로 테스트케이스의 개수도 많고 테스트케이스를 구성하는 항목도 많으므로 입력해야 할 데이터도 많다. 입력데이터를 대량으로 조합하는 Pair-wise 기법을 사용할 수 있으나 실험의 결과로 복잡한 엔터프라이즈 환경에서는 조합 가능한 단위로 개발자가 입력하여 가능한 모든 조합을 이용하는 것이 효과적이다. 테스트 데이터를 단순하게 입력하는 것은 은행 어플리케이션에서는 사실상 불가능하다. 테스트 데이터를 입력하기 위해서는 Record & Replay 방식에서 Record의 기능에 해당하는 로그 참조에 의한 등록을 제안한다. 이렇게 하면 예상결과를 입력할 때 모든 필드 값을 계산하거나 추축할 필요가 없어진다. 테스트 도구에서 테스트 실행은 One-Click으로 하고 테스트 실행 후 DB의 상태를 유지하기 위해 완료 (Commit), 취소 (Rollback)를 선택할 수 있는 수단을 제공하고 개발서버와 테스트 서버를 정하여 실행할 수 있는 수단이 필요하다. 테스트 결과는 DB와 로그로 관리되어 투명하게 실시간으로 현황을 파악할 수 있는 수단을 제공해야 한다. 은행 어플리케이션은 복잡한 비즈니스와 정보기술로 이루어짐으로 환경 변수에 의한 조정이 될 수 있게 하여 테스트 도구의 복잡성을 줄여야 한다.

은행 테스트 대상 컴포넌트는 소스와 서비스(또는 서버)로 구성된다. 따라서, 테스트 도구는 테스트케이스 데이터를 이용하여 드라이버 소스를 생성하여 테스트 대상 소스 프로그램을 테스트하는 방식과 미들웨어 기반에서 실행 대기 중인 서비스(또는 서버)에게 입출력 전문을 주고 받으며 테스트할 수 있는 수단이 제공되어야 한다. eTDD에서 테스트 도구는 복잡한 비즈니스 환경을 수용하기 위해 데이터베이스를 이용하여 한다. XUnit과 같이 소스코드로 작성하는 데에는 많은 시간과 공수가 소요된다.

마지막으로 eTDD는 통합된 테스트 진척을 관리하기 위해 프로젝트관리 시스템과 연계, 테스트 대상 소스코드의 버전을 관리하기 위해 형상관리 시스템과 연계, 회귀 대상을 추출하기 위해 영향도 분석 시스템과 연계, 개발 프로세스 상에 끼워 넣기 위해 전자 결재를 지원하기 위해 그룹웨어와 연계 등과 같은 외부 시스템과 연계할 수 있는 수단이 필요하다.

## 4. eTDD 지원도구

### 4.1 개념 모델

eTDD 지원도구는 IEEE 829 (1998, p.7)에 따라 테스트케이스는 대상항목, 입력, 출력, 환경, 수행절차, 테스트케이스간 연관관계를 모델링하였다. eTDD 개념 모델은 그림 2와 같은 개념의 관계로 나타낼 수 있고, 테스트슈트를 비즈니스와 연관 짓기 위해 업무와 프로그램 개념을 연결하였다. 테스트 슈트는 개별 함수와 동일하며, 개발자는 이 함수를 테스트하기 위해 다양한 입력값과 예상값을 작성한다. 함수 호출시 보내고 받는 입력값과 예상값의 집합을 테스트케이스라고 한다. 테스트케이스는 입력데이터, 예상데이터, DB입력데이터, DB예상데이터로 구성된다. 테스트를 실행하면 결과를 담기 위해 실행 개념을 도입한다. 실행한 결과는 로그로 남고 이를 이용해야 하므로 로그 개념을 포함하였다.

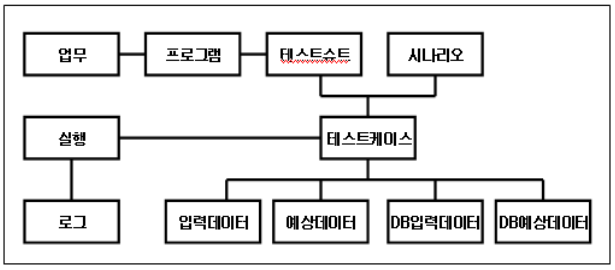


**그림 2 – eTDD 개념 모델**

### 4.2 구현

앞서 제안된 eTDD 프레임워크는 그림4의 아키텍처로 구현되었다. 테스터는 비즈니스 실행을 위한 Client 단말과 테스트 수행을 위해 Web Browser를 이용한다. Client 단말은 SUT(System Under Test)와 직접 화면에서 입· 출력 데이터를 넣고 기능을 수행하고, 실행된 서비스는 디버그 로그를 파일로 남기며, 요청과 응답의 결과인 입· 출력 전문을 참조에서 활용하기 위해 DB 테이블에 저장한다. Web Browser와 연결되는 테스트 도구의 중요 컴포넌트는 그림 3의 점선 안에 있는 내용과 같다. eTDD 기능별로 보면 (1) 테스트케이스 입력은 Test case Manager, (2) 테스트케이스 실행에는 Test

Executor, Interface Manager, Test Result Manager 등, (3) 테스트 관리에 필요한 Defect Manager, Progress manager등이 구현되었다. (4) 테스트의 전사적 실행과 반복실행을 위해 테스트케이스는 RDB로 저장, 관리되도록 구현하였다. 테스트를 수행한 결과인 테스트 로그를 파일과 DB 테이블에 저장하여 테스트 실행 결과의 추적이 용이하게 한다.

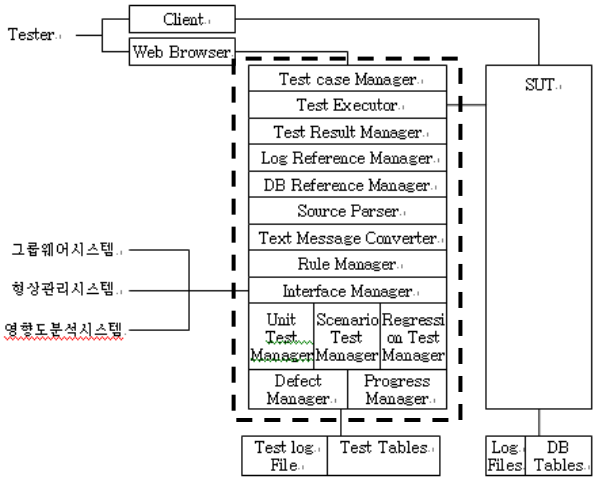


**그림 3 – eTDD 아키텍처**

## 5. eTDD의 적용과 시사점

eTDD를 은행의 차세대 프로젝트에 장기간 적용하면서 얻은 시사점은 다음과 같다.

(1) 테스트케이스 생성의 어려움: 테스트케이스는 대상함수, 입력값과 예상결과값으로 구성한다 (IEEE 620). 본 사례에서는 카테고리선택에 따라 가능한 경로를 테스트케이스로 작성하여 블랙박스 (또는 그레이박스) 방식으로 시행되었다. 그러나 앞서 논의한 바와 같이 은행어플리케이션은 그 특성상 입· 출력 파라미터의 개수가 많고, 함수 처리 로직이 복잡하여 테스트케이스의 수작업은 불가능에 가깝다. 업무에 따라 차이가 있었지만 수백 개의 출력 파라미터에 대한 예상결과값을 작성하는 사례가 많아 단순한 수작업으로는 실용적이지 않고 거의 불가능에 가까웠다. 입, 출력 값의 예를 들면, 수신업무는 적은 경향이 있었고, 반면 여신, 외국환, 고객 등에는 매우 많았다. 또한 1본의 프로그램이 적게는 수백에서 수만 줄 (Lines of Code)에 달했다. 이러한 어려움이 개발자로 하여금 테스트케이스의 적용에 대한 거부감으로 나타난 경향이 있었다고 하겠다.

(2) 테스트 프로세스의 차이: 본 사례에서 개발프로세스를 관찰하면 '테스트케이스가 개발 이전에 정의될 수 있을까?' 하는 의문이 있다. 즉 개발자는 상세설계를 개발과 더불어 수행하는 경향이 있었다. 따라서 개발이 끝나고 나서야 함수가 안정화되고 비로서 테스트케이스를 정의하는 행태가 있었다. 이는 TDD의 목적과 정면으로 배치되는 작업절차가 아닐 수 없다. 개발자는 우선 임의적인 테스트방식으로 개발, 테스트, 디버깅을 수행하고, 결함이 상당한 수준으로 줄어들면 비로서 TDD방식을 따르는 이중적인 태도를 보였다.

(3) 테스트케이스의 수: 테스트케이스는 쉽게 그 수가 늘어 날 수 있다. 은행어플리케이션은 입력 파라미터의 개수와 각 파라미터의 가능 값이 많아 폭발적으로 증가하였다 ($(v/2)\log_n k$ [25]). 사례에서는 공통 모듈에 대한 테스트케이스는 대략 수십 개를 작성하였으며, 정상적 로직의 검증에 초점을 맞추었다. 그 외의 테스트케이스는 서비스를 통해서 호출되고 처리되는 경로를 점검하면서 테스트하였다. 작성된 테스트케이스는 해당 테스트 단위 별 보통 20여 개에서 최대는 1023개에 달했다.

(4) 테스트케이스 재사용: 사례에서는 단위함수에 대해 테스트가 완료되면 요구사항의 수정에 따라 프로그램이 변경되어도 추가적인 공수의 투입에 대한 거부감으로 테스트의 반복에 거부감이 있었다. 이러한 태도는 TDD의 적용을 어렵게 할 것이며 본 사례에서 개발된 지원도구와 같은 테스트환경의 필요성을 강조하고 있다.

(5) 테스트 환경의 복잡성: 테스트의 전사적 적용을 어렵게 하는 요인으로는 테스트의 실행 환경이 매우 복잡하다는 것이다. 예를 들면, 테스트 대상이 소스 코드인 경우 컴파일, 링크의 환경은 복잡하겠지만, 그 대상이 공유 라이브러리 또는 미들웨어 서비스 형태인 경우는 어려움은 줄어들 수 있다. 어플리케이션 프레임웍을 사용하는 경우에는 공유라이브러리 형태로 테스트하는 경우에는 프레임웍 의존부분이 있는데 이것이 실행에서 고려해야 할 사항이다. 테스트 실행의 결과는 즉시 확인할 수 있도록 온라인으로 구성해야 한다. 테스트 이해당사자들은 테스트를 실행한 결과를 항상 동일하게 볼 수 있도록 구성해야 한다. 특히, 테스트 실행에서의 중요한 이슈 중에 하나는 개발자가 테스트한 시점에는 오류가 없었는데 테스트 프로세스를 거치는 중에 공통 모듈이 변경되어 이 영향으로 오류가 발생할 수 있다. 이러한 이슈를 해결하기 위해 테스트 시점의 결과를 저장할 수 있는 수단을 제공해야 한다. 사례의 은행 어플리케이션 테스트에서는 개발서버의 환경과 테스트서버의 환경을 (소스, 바이너리, 거래DB 등)을 일치화 시키기 어려워 하나의 서버 합쳐서 테스트를 수행하였다.

(6) 테스트실행 자동화: 개발자는 엑셀의 편집기능을 선호하여 이를 이용하여 테스트케이스를 관리하려는 경향이 있었으나 테스트 실행은 자동화하여 수행 될 수 있는 기능 원하였다. 또한 수신 어플리케이션을 예로 들면, 신규 통장을

개설하는 업무, 이 통장에 입금하는 업무, 이 통장에서 출금하는 업무, 마지막으로 이 통장이 필요 없으면 해지하는 업무를 순서대로 테스트를 할 수 있는 기능 요구하였다. 테스트 대상 소스 버전 관리를 위하여 형상관리 시스템과 연계하고, 공통 모듈이 수정될 경우 회귀 분석 저장을 하기 위해 영향도 분석 시스템과 연계에 대한 요구가 있었다. 그리고, 통합 테스트와 지점 테스트에서 발생한 거래를 이용하여 테스트를 재수행하거나 기존 거래를 활용하는 요구가 있었다.

(7) 테스트데이터의 유지 어려움: 은행 어플리케이션의 함수 입· 출력 파라미터와 서비스 입· 출력 전문은 프로젝트의 종료 시점까지도 변경되는 특징이 있었다. 이러한 변경은 구조체 멤버를 추가하거나 삭제하고, 이 멤버의 변경 결과는 이미 등록된 값 일관성을 유지하기 위해 동기화해야 하였다. 또한 함수의 처리 로직이 변경되면 테스트케이스의 값도 변경해야 하는 문제가 있었다. 이러한 변경에 강력하게 대처하기 위해서는 입력 및 예상 데이터에 룰 (예를 들면, 예상의 잔액은 = 입력의 입금액 + 입력의 잔액)을 지원하여 테스트가 수행되어도 테스트 실행이 정상적으로 동작하도록 하였다. 테스트 데이터를 유지하기 위한 또 다른 방안으로는 별도의 테스트 서버를 구축하고, 테스트완료 후 취소(Rollback)하여 테스트 데이터가 변경되지 않도록 하였다.

(8) DB기반 테스트의 어려움: 은행 어플리케이션 또는 전사적인 시스템은 DB를 기반으로 개발되어 단위테스트는 DB를 접근하여 확인하는 번거로움이 수반된다. 즉, 개발자는 테스트가 성공하였다는 증빙으로 테이블을 조회하여 하드카피로 출력하여 제출하는 것이 요구되었다. eTDD 지원도구는 테스트의 결과를 조회할 수 있는 기능이 제공되었으며, 테이블의 필드가 많은 경우에 오류에 국한하여 조회할 수 있도록 하였다. 이는 XUnit과 같은 종전의 TDD도구에서 단위함수에 초점을 맞춘 것과 달리 eTDD에서는 DB연결, 조회 관련 지원이 필요함을 말해 주고 있다.

(9) 행태적 저항: 개발자는 테스트 자체를 싫어하는 경향이 관찰되었고, 개발자는 도구의 사용이 불편하고, 테스트케이스 작성에 공수가 많이 소요된다는 등의 이유를 들어 옛날 테스트 방식을 고집하는 등 TDD에 부정적인 행태가 발견되었다. 개발자는 대부분 테스트를 제대로 하지 않기 때문에 테스트를 제대로 하는데 소요되는 노력과 비교하는 것으로는 개발자를 설득하기 어려웠다. 프로젝트관리자는 eTDD가 품질을 향상하고, 전통적인 개발에서 볼 수 있는 통합의 어려움을 줄일 수 있다는 점에서 접근하였고, 개발자는 자신의 개발프로세스와 배치된다는 점에서 적용에 거부감을 보였다.

## 6. 결론과 향후 연구과제

최근 TDD의 효과성에 대한 평가가 이루어지면서 전사적인 적용의 필요성이 제기되고 있지만, 대규모 프로젝트에서 수행된 사례가 적어서 실무자에게 도움이 되지 않고 있다. 본 연구에서는 eTDD를 TDD의 전사적 적용이라고 정의하고 프레임워크를 제안하였고, 이를 기반으로 지원도구를 구축, 대형 은행 차세대 프로젝트에 적용하였다. 그 결과 TDD를 차세대와 같은 대규모의 프로젝트에 적용하기 위해서는 선결과제가 많은 것으로 나타났다. 은행은 실시간 운용환경의 특성과 더불어 비즈니스 로직과 이를 구현한 함수의 호출 구조가 순환, 계층적으로 구성되어 테스트의 어려움을 가중하고 있어 이에 대한 관련 연구가 시급하다. 본 연구에서는 외환 환전 프로그램의 경우 1023개 테스트케이스 실행하여 찾기 매우 어려운 메모리의 2개 결함을 발견하였다. 그러나 TDD의 전사적 적용을 위해서는 이러한 유용성에도 불구하고 사용성에 대한 연구가 필요하다. Davis 의 TAM모델에서는 효과성과 더불어 사용의 편리성이 기술적 도구의 사용을 결정하는 과정에 중요한 변수가 된다고 강조하였다 [24]. TDD는 테스트의 중요성을 감안할 경우 전사적으로 적용되는 것이 필요하지만, 관련한 테스트의 지원이 필수적이며 그 사용이 매우 편리해야 함을 사례에서 보여주고 있다.

특히 복잡한 비즈니스를 테스트하기 위해서는 테스트 자동화 지원하는 도구가 필요함에도 불구하고 많은 개발자들은 강하게 저항하였다. 테스트케이스 작성에 과다한 노력이 투입된다는 것을 빌미로 자신들이 수행한 테스트 작업이 공개되는 것을 꺼려하는 경향이 있었다. 개발자는 습관적으로 개발을 먼저 함으로 테스트케이스가 개발에 선행하는 프로세스를 수용하지 않는 경향이 있었다. TDD의 전사적인 적용을 위해서는 관련도구의 개발과 더불어 품질과 생산성으로 국한하기 보다는 이러한 개발자의 행태적인 측면에 대한 연구가 필요하다. 본 연구는 유례없이 큰 규모로 수행된 사례를 대상으로 테스트한 경험을 보여주고 있어, 향후 관련 연구에 많은 과제와 시사점을 제공하고 있다.

## References


[1] Beck, K. (2003). "*Test-Driven Development: By Example*". Boston, MA.: Addison-Wesley.

[2] Bhat, T. & Nagappan, N., *Evaluating the efficacy of test-driven development: industrial case studies*, in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. 2006, ACM: Rio de Janeiro, Brazil.

[3] Desai, H.D. (1994). *Test Case Management System*

*(TCMS)*. in *Global Telecommunications Conference, 1994. GLOBECOM '94. Communications: The Global Bridge., IEEE*. pp. 1581-1585.

[4] Erdogmus, H., Morisio, M. & Torchiano, M. (2005). "On the effectiveness of the test-first approach to programming." *IEEE Transactions on Software Engineering*, **31** (3) pp. 226-237.

[5] George, B. & Williams, L. (2004). "A structured experiment of test-driven development." *Information and Software Technology*, **46** (5) pp. 337-342.

[6] Geras, A.M., Smith, M.R. & Miller, J. (2004). "A survey of software testing practices in Alberta." *Canadian Journal of Electrical and Computer Engineering*, **29** (3) pp. 183-191.

[7] Giraudo, G. & Tonella, P. (2003). "Designing and conducting an empirical study on test management automation." *Empirical Software Engineering*, **8** (1) pp. 59-81.

[8] Ho-Yeon, R., Byeong-Kil, S. & Jae-Heung, P. (2005). "Mock objects framework for TDD in the network environment." *Proceedings. Fourth Annual ACIS International Conference on Computer and Information Science|Proceedings. Fourth Annual ACIS International Conference on Computer and Information Science*, pp. 430-4|xv+719.

[9] Janzen, D. & Saiedian, H. (2005). "Test-driven development: Concepts, taxonomy, and future direction." *Computer*, **38** (9) pp. 43-+.

[10] Janzen, D.S. & Saiedian, H. (2006). "On the influence of test-driven development on software design." *Proceedings. 19th Conference on Software Engineering Education & Training|Proceedings. 19th Conference on Software Engineering Education & Training*, pp. 8 pp.|CD-ROM.

[11] La Commare, G., Giraudo, G. & Tonella, P. (2000). *Test management automation: Lessons learned from a process improvement experiment*, in *Software Process Technology*. Springer-Verlag Berlin: Berlin. pp. 156-160.

[12] Larman, C. & Basili, V.R. (2003). "Iterative and incremental development: A brief history." *Computer*, **36** (6) pp. 47-+.

[13] Lulu, L. & Robson, D.J. (1992). *SEMST-a support environment for the management of software testing*. in *Assessment of Quality Software Development Tools, 1992., Proceedings of the Second Symposium on*. pp. 11-20.

[14] Maximilien, E.M. & Williams, L. (2003). "Assessing test-driven development at IBM." *25th International Conference on Software Engineering, Proceedings.*, pp. 564-569.

[15] Mukkamala, R., Pedagani, R. & Keskar, H. (2005). "TDMS for aviation software." *IEEE Aerospace and Electronic Systems Magazine*, **20** (12) pp. 15-20.

[16] Murray, L., Carrington, D., MacColl, I. & Strooper, P. (1999). *TinMan-a test derivation and management tool for specification-based class testing*. in *Technology of Object-Oriented Languages and Systems, 1999. TOOLS 32. Proceedings*. pp. 222-233.

[17] Sangwan, R.S. & LaPlante, P.A. (2006). "Test-driven development in large projects." *IT Professional*, **8** (5) pp. 25-9.

[18] Williams, L., Maximilien, E.M. & Vouk, M. (2003). "Test-driven development as a defect-reduction practice." 14th International Symposium on Software Reliability Engineering, pp. 34-45.

[19] Woodward, M.R. & Hennell, M.A. (2005). "Strategic benefits of software test management: a case study." *Journal of Engineering and Technology Management*, **22** (1-2) pp. 113-140.

[20] Canfora, G., Cimitile, A., Garcia, F., Piattini, M. & Visaggio, C. A. (2006). "Productivity of test driven development: a controlled experiment with professionals." Product-Focused Software Process Improvement. 7th International Conference, PROFES 2006. Proceedings (Lecture Notes in Computer Science Vol. 4034): pp.383-8.

[21] Edwards, S. H. (2003). Using Test-Driven Development in the Classroom: Providing Students with Automatic, Concrete Feedback on Performance. International Conference on Education and Information Systems: Technologies and Applications (EISTA03), Orlando, Florida, USA.

[22] Muller, M. M. & Hagner, O. (2002). "Experiment about test-first programming." Software, IEE Proceedings - 149(5): pp.131-136.

[23] Pancur, M., Ciglaric, M., Trampus, M. & Vidmar, T. (2003). "Towards empirical evaluation of test-driven development in a university environment." IEEE Region 8 EUROCON 2003. Computer as a Tool. Proceedings (Cat. No.03EX655): pp.83-6.

[24] Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Quarterly, 13(3), pp.319-340.

[25] Cohen, D.M. Dalal, S.R. Fredman, M.L. & Patton, G.C. (1997) The AETG System: An Approach to Testing Based on Combinatorial Design. IEEE Transactions on Software Engineering, 23(7): pp.437-444.