코드 커버리지 테스팅 기법의 금융권 차세대 프로젝트 적용사례

김희영*°, 양상태**
*SK C&C IT공학센터 품질혁신팀, 한국외국어대학교 경영학과 박사과정,
**SK C&C 공공/금융사업부문 Delivery혁신센터

Code coverage Testing in Next Generation Banking System Project

Kim, Hee Yeong°, Yang, Sang Tae**
SK C&C IT Engineering Center, SK C&C Delivery Innovation Center
E-mail : heeykim@skcc.com, styang@skcc.com

요 약

대규모 금융권 차세대 프로젝트에서 테스트에 대한 중요성은 재무RISK 관점에서 뿐만 아니라 소프트웨어 품질향상을 위한 결함의 제거 관점에서도 그 의미가 크다. 대규모 차세대 프로젝트는 일반 프로젝트에 비하여 개발되는 소프트웨어의 양이 방대하고 개발참여인원도 일반적인 관리수준을 넘어 수작업에 의한 테스트만으로는 충분한 품질을 보장하지 못한다. 또한 테스트를 수행한 이후에도 지속적으로 변경이 발생하고 이를 반영하는 과정에서 많은 결함이 유입되는 현재의 SI 프로젝트 특성상 지속적인 테스트 및 반복적인 검증만이 소프트웨어의 품질을 보장할 수 있다. 본 논문에서는 code coverage testing기법을 활용하여 동적 테스트 수행결과를 log로 도출하고, log 분석 결과를 통해 소프트웨어 품질의 향상을 기대할 수 있는 방안을 제시한다.

1. 서론

테스트를 수작업으로 처리한다는 것은 매우 복잡하고 관리상 한계가 있기 때문에 테스트 작업을 편리하게 수행하기 위해 많은 자동화 도구가 개발되고 있다. 특히 최근 대형 SI 프로젝트 특성상 온라인과 배치 프로그램, 각종 툴이 연계되어 수행되는 경우, 전체 테스트 범위 및 수준을 파악할 수 있는 최적의 툴이 아직 충분하지 않다.

또한 컴포넌트 기반의 개발 프로세스로 진행하는 금융권 시스템 특유의 업무 성격상 객관적인 테스팅 현황을 파악하는데 비 전문가의 테스트 결과만을 믿고 테스트단계를 종료처리 한다면 문제가 있다.

본 논문에서는 컴포넌트 기반 개발 프로세스 성격

의 금융권 시스템에서 코드 커버리지 기법을 적용한 소프트웨어 검증을 통해서 테스트의 효과를 극대화 할 수 있는 적용방안을 제시하고자 한다.
이를 통해 기대되는 효과로는 각 업무 시스템별 테스트케이스에 누락된 소스코드 영역을 파악하고 이를 개선함으로써 테스트의 충실도를 향상시키고 리스크(Risk) 감소 및 안정적이고 효율적인 시스템을 구축할 수 있다는 확신을 의사결정권자에게 객관적 수치로서 보여 줄 수 있다.

.

2. 소프트웨어 테스팅 기법

2.1 테스팅 기법의 종류

소프트웨어 테스팅은 프로그램을 실행시키지 않고 테스팅을 수행하는 정적 테스팅(Static Testing)과 프로그램을 실행시켜 테스팅을 수행하는 동적 테스팅(Dynamic Testing)으로 분류할 수 있다.

동적 테스팅은 다시 블랙박스 테스팅(black-box Testing)과 화이트박스 테스팅(white-box Testing)으로 나눌 수 있다. 블랙박스 테스팅은 기능적 테스트라고도 하며, 프로그램의 구조를 고려하지 않고 프로그램의 요구사항 명세로부터 테스트 데이터를 선정한다. 반면 화이트박스 테스팅은 구조적 테스팅이라고 하며, 구현된 프로그램 내부 구조를 보고 테스트 데이터를 선정한다.
구조적 테스팅 기법은 특정 유형의 구조 커버리지를 평가하여 테스팅의 보장성 또는 충분함(Thoroughness)을 측정하는 것이므로 이를 위해서는 명세기반 기법을 적용 후 사용할 때 가장 좋은 결과를 보여준다.

2.2 코드 커버리지 테스팅 기법

코드 커버리지(Code Coverage)는 구조적 테스팅 기법의 하나로 주어진 테스트케이스에 의해 수행되는 테스트의 범위를 측정하는 기준이며, 소스 코드의 분명한 목적이 테스트 프로그램에 행해지고 있는지 비교한다. 이것은 프로그램 작동 및 프로그램 구조와 로직에서의 문제점을 테스팅한다. 또한 구조적 커버리지 분석을 통해 의도하지 않는 기능 유무 및 테스트 케이스의 적절성을 판단할 수도 있다.
코드 커버리지는 소프트웨어 테스트의 품질을 측정하는 도구 중 하나로, 말 그대로 현재 테스트 코드가 프로젝트 코드를 얼마나 커버하고 있는지를 수치로 알려준다.
일반적으로 코드 커버리지 분석은 아래와 같은 목적으로 이루어진다[1].

- 테스트케이스 집합에 의해 수행되지 않는 프로그램 영역을 찾는다.
- 커버리지를 증가시키기 위해서 추가적인 테스트케이스들을 생성한다.

코드 커버리지의 궁극적인 목표는 프로그램 내부 구조에 존재하는 모든 경로를 실행하는 것이지만 경로의 수가 대단히 많아서 이를 모두 시험하는 것은 불가능하다. 그렇다면 합리적인 방법으로 커버리지를 선정하는 기준이 필요한데 원자력발전이나 항공기 관련 소프트웨어는 적용사례가 있으나 금융권 프로젝트에서 현재까지 그 특성에 맞는 커버리지 적용기준이 없는 형편이다.

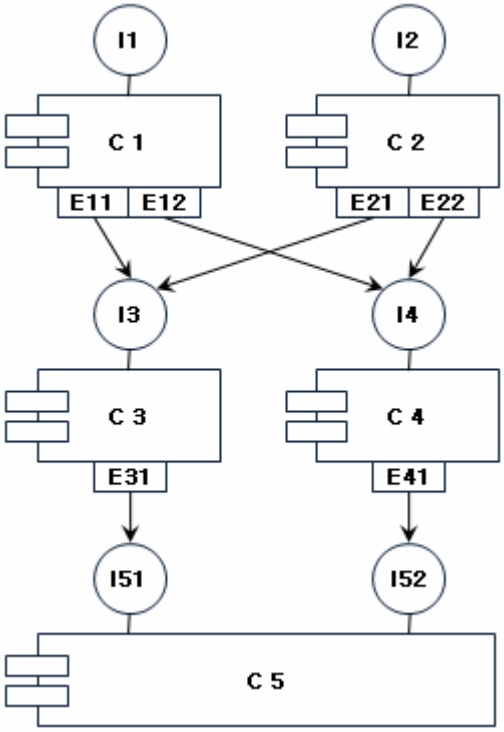
2.3 컴포넌트 기반 소프트웨어 테스팅

컴포넌트는 독립적인 개발, 배포, 조립이 가능한 소프트웨어의 구성 단위이다. 재사용이 높은 컴포넌트를 조립하는 방식으로 대규모 소프트웨어 시스템을 개발함으로써 기존의 방식에 비하여 보다 빠른 시간에 보다 높은 품질의 시스템을 구축할 수 있다는 이점이 있다[2].
기존의 컴포넌트들을 조립하는 방법을 이용하여 시스템을 개발할 때는 당연히 조립되는 개별 컴포

넌트들에 대하여 지금까지 소개한 블랙 박스 테스팅과 화이트 박스 테스팅 기법을 적용하여 테스팅을 수행해야 한다. 뿐만 아니라, 각 컴포넌트는 다른 개발자, 다른 프로그래밍 언어, 다른 운영 환경에서 개발되었기 때문에 각 컴포넌트들이 정확하게 상호작용을 하는지에 대한 확인을 위하여 컴포넌트 통합 테스팅도 수행할 필요가 있다.
따라서, 컴포넌트 기반 소프트웨어서는 개별적인 컴포넌트에 대한 테스팅 뿐만 아니라, 이들 컴포넌트들이 통합되어 시스템 관점에서 올바르게 동작을 할 수 있는 지에 대한 확인이 매우 중요한 요소로서 작용하게 된다[3].

컴포넌트 간의 상호 작용을 테스팅하기 위하여 테스팅 될 컴포넌트의 상호 작용에 대한 정보를 명확하게 기술할 필요가 있다. 예를 들어, [그림 1]은 5개의 컴포넌트 간의 상호 작용을 보여 주고 있다. [그림 1] 에는 C1, C2..... C5 이름의 5개의 컴포넌트가 있으며, 각 컴포넌트는 I1, I2..... I51, I52 이름의 6개의 인터페이스를 각각 가지고 있다. 컴포넌트 하단에 위치한 E11, E12 등은 해당 컴포넌트가 인터페이스를 통하여 다른 컴포넌트를 호출하는 것을 의미한다. 즉, ---> 는 한 컴포넌트와 다른 컴포넌트 간의 호출 관계에 의한 상호 작용을 뜻한다. 예를 들어, C1 컴포넌트는 E11 위치에서 I3 인터페이스를 통하여 C3 컴포넌트를 호출하고, E12 위치에서 I4 인터페이스를 통하여 C4 컴포넌트를 호출하고 있다.

[그림 1]과 같이 컴포넌트 간의 상호 작용이 정의되었다면 이를 바탕으로 다양한 수준으로 컴포넌트 간의 상호 작용에 대한 테스팅 기법들을 적용할 수 있다[2]. 이와 같이 소프트웨어 개발 과정에서 소프트웨어가 가지고 있는 결함을 발견해내는 테스트는 품질 보증이나 검증 면에서 상당히 중요한 위치를 차지한다. 총괄적인 의미에서 “테스트는 에러를 발견하는 것을 목적으로 하여 프로그램을 실행하는 과정이다”라고 정의할 수 있다[4].



[그림 1] 컴포넌트들 간의 상호 작용

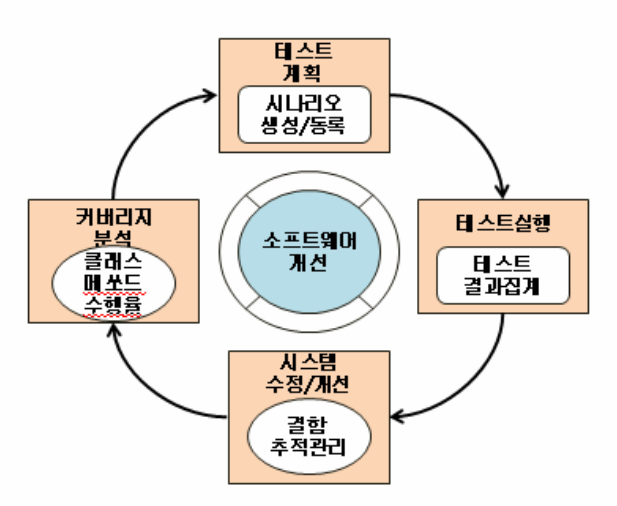
그러나 소프트웨어의 모든 동작 환경을 있는 대로 실행해주는 테스트는 일반적으로 불가능하고 실현성도 없다. 그래서 보편적으로 부족한 상태에서 테스트를 실시하여 그 결과를 바탕으로 그 제품이 실용 레벨에 도달했는지 여부를 평가, 판정하지만 완전한 해결 방법이 되지 못하는 실정이다. 또한, 얼마나 많은 잠재적인 결함이 테스트 집합에 의해 발견되는지를 정량적으로 제시하는 것은 필수적이며, 여기에 소요되는 시간은 가능한 한 짧으면 좋다.

3. 컴포넌트 기반 커버리지 분석

3.1 커버리지 분석 테스팅 사이클

각종 시스템과 레포팅 툴들로 연계되는 대형 SI 프로젝트에서 테스팅 단계의 효과를 높이려는 노

력은 무엇보다 중요하다. 또한 컴포넌트 기반의 프로젝트에서는 시스템간의 인터페이스 테스트를 위해 외부로 노출시킨 컴포넌트까지 테스트가 가능한 통합테스트 단계가 커버리지 적용을 통한 효과를 극대화 할 수 있는 단계라 할 수 있다.



[그림 2] 커버리지 분석 테스팅 사이클

[그림 2]는 커버리지 분석을 통한 테스팅 사이클을 보여주고 있는데 테스트케이스 수행을 통한 테스팅으로 결함이 발견되면 결함관리시스템(*결함을 등록, 추적, 관리하는 시스템)에 결함을 등록하게 된다. 물론 결함조치는 결함관리시스템에서 이력관리 된다. 그러나 테스트케이스 수행만으로 시스템을 개선하기에는 한계가 있다. 테스트케이스가 전체 시스템에 어느 정도 범위를 커버하는지, 실제로 중요한 컴포넌트는 테스트가 되었고 오류는 없었는지를 제대로 파악하지 못한다면 테스트의 신뢰성은 떨어질 수 밖에 없을 것이다. 이러한 한계성을 극복하기 위해서는 잠재된 결함의 가능성을 낮추고 사용자의 신뢰성을 높이기 위한 방안으로 로그분석을 통한 효과적인 커버리지 분석 전략이 무엇보다 필요한 것이다.

또한 커버리지 분석을 위해서는 테스트 시 생성되는 로그정보를 분석하여 각 클래스 단위의 메쏘드 수행 정보를 통해 테스트케이스 커버리지를 측정하게 되는데 그 적용대상은 J2EE 프레임워크 기반의 전체 어플리케이션 (온라인, 배치 컴포넌트)이며 J2EE 프레임워크에서 생성되는 정형화된 로그정보를 코드 커버리지 측정 데이터로 활용한다. 기존 로그 분석 툴들은 배치처리 및 다양한 개발언어를 동시에 분석하는 것은 지원하지 않고 있어서 J2EE 프레임워크에서 일정한 포맷에 따라 로그를 생성토록 하고 분석을 위한 추출파일을 데이터베이스에 입력하여 커버리지를 산출하게 한다.

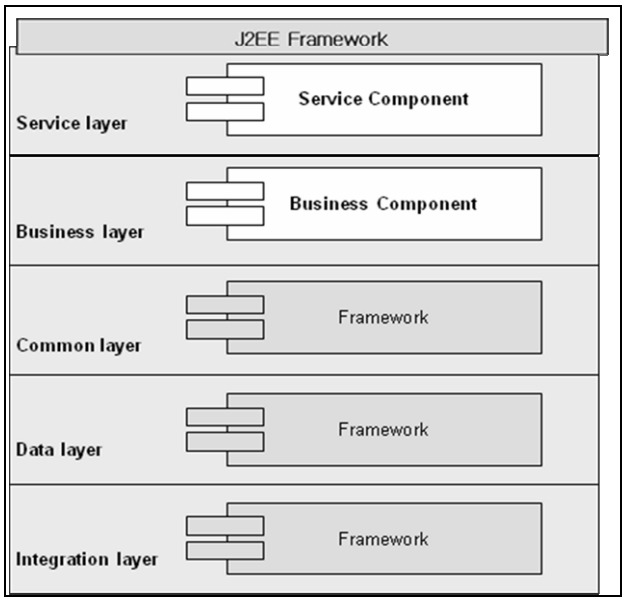
3.2 Function 커버리지 측정

대형 SI 프로젝트 특성을 고려하여 커버리지는 Function 커버리지 기법으로 진행한다. 원자력발전이나 항공기 관련 소프트웨어가 안전성에 중점을 두어 레벨에 따른 커버리지를 규정해 둔 반면, 일반 SI 프로젝트는 각각의 시스템간 통합 및 각종 레포팅 툴과의 연계 작업 및 개발 언어의 다양화로 각종 표준 인터페이스가 존재하게 된다. 이런 환경의 컴포넌트 기반 프로젝트에서는 컴포넌트 자체 문장 보다는 컴포넌트 상호작용 테스팅에 훨씬 더 비중이 크다.

시간과 비용 그리고 인력이 충분하지 않는 대형 SI 프로젝트에서 주요 기능 등에 대한 메쏘드와 클래스의 커버리지 목표 달성으로도 기존 테스팅 전략의 효과를 증대 시킬 수 있다.

3.3 컴포넌트 세분화

일반적으로 J2EE 프레임워크 기반에서 작동하는 컴포넌트들을 식별하여 세분화하는 커버리지 분석 전략이 테스팅 효과를 극대화 할 수 있다. 여기에서 프레임워크이란, 업무 어플리케이션을 쉽게 구축할 수 있도록 사전에 미리 준비된 작업환경으로써 이를 이용하게 되면 개발 표준화 및 재사용을 통해 생산성 및 시스템 안정성을 높일 수 있게 된다.



[그림 3] J2EE 프레임워크 계층 구조

[그림 3]은 J2EE 프레임워크 계층을 나타내고 있는데, 최상위부터 서비스 계층, 비즈니스 계층, 공통 계층, 데이터 계층, 통합 계층으로 나뉜다.

[그림 3]의 J2EE 프레임워크 계층에서 function 커버리지 대상은 실제 개발자가 구현하는 서비스 영역의 서비스 컴포넌트, 비즈니스 영역의 비즈니스 컴포넌트이다. 또한 컴포넌트를 구성하는 클래스와 메쏘드까지 세분화하여 커버리지 분석을 하였다. 아래 [표 1]은 Function 커버리지를 기능, 서비스 컴포넌트, 비즈니스 컴포넌트, 클래스, 메쏘드, 문장 등 6단계 분류하였는데 본 논문에서 5 레벨 수준까지 분석하도록 하였다.

[표 1] 커버리지 granularity 테스팅 레벨

레벨	granularity	상세
1	기능 (function)	컴포넌트의 집합으로 시스템의 작은 기능의 수행 여부를 확인할 수 있는 수준
2	서비스 컴포넌트 (Service Component)	서비스의 처리흐름을 관리하는 컴포넌트 J2EE 프레임워크 패키지 계층구조의 서비스 task
3	비즈니스 컴포넌트 (Business Component)	단위업무처리를 담당 데이터의 생성, 조회, 수정, 삭제 처리 및 계산 J2EE 프레임워크 패키지 계층구조의 비즈니스 task
4	클래스 (Class)	메쏘드의 집합으로 컴포넌트를 구성하는 프로그램 단위
5	메쏘드 (Method)	프로그램을 구성하는 문장(statement)의 집합
6	문장 (Statement)	Java 소스코드의 절 실제 수행을 위한 분기 및 결정단위

여기서 granularity는 테스팅이 커버하게 되는 작은 프로그램 모듈의 수준을 말하며, 세부적인 테스트까지 가능하도록 granularity 수준을 높이면 시간과 노력이 훨씬 많이 소모되나, 충분히 많은 경우의 수의 테스트 케이스를 커버하게 된다.

[표 1]에서 레벨 6 수준으로 테스트를 수행하기 위해서는 테스트 케이스/시나리오가 거의 프로그램 수준이 되게 되며, 소스코드를 직접 읽어서 테스트를 수행하는 것과 차이가 없게 된다. 이러한 경우에는 소스 코드 인스펙션을 수행하는 것이 더 효과적일 수 있다.

3.4 가중치에 따른 중요도 분류

테스팅 대상을 업무의 비중, 수행 빈도수, critical 여부에 따라 점수를 부여하고 중요도를 상/중/하로 분류하였다.

중요도 판정에 있어서 critical하다는 것은 금전적인 처리를 위한 부분이나, 신용등급 판단에 필요한 부분 등이 되며, 업무비중은 해당 컴포넌트의 업무에 차지하는 영역이 조직적인 측면에서 차지하는 위치를 말한다. 그리고 각각의 구성요소는 1 ~5 점을 부여하며, 전체 중요도는 '상'은 11 ~ 15점, '중'은 6 ~ 10점, '하'는 0 ~ 5점으로 분류하여 측정하도록 하였다.

[표 2]은 테스팅 수행단위를 레벨 3수준으로 결정하였을 경우에 중요도 판단의 단위가 비즈니스 컴포넌트임을 가정하고 중요도 판단을 위한 표를 예로 나타낸 것이다.

[표 2] 커버리지 granularity 적용(예)

테스팅 단위	업무 비중	수행 빈도	Critical	총점	중요도
비즈니스 컴포넌트 001	5	3	3	11	상
비즈니스 컴포넌트 002	1	3	3	7	중
비즈니스 컴포넌트 003	1	1	3	5	하
……					

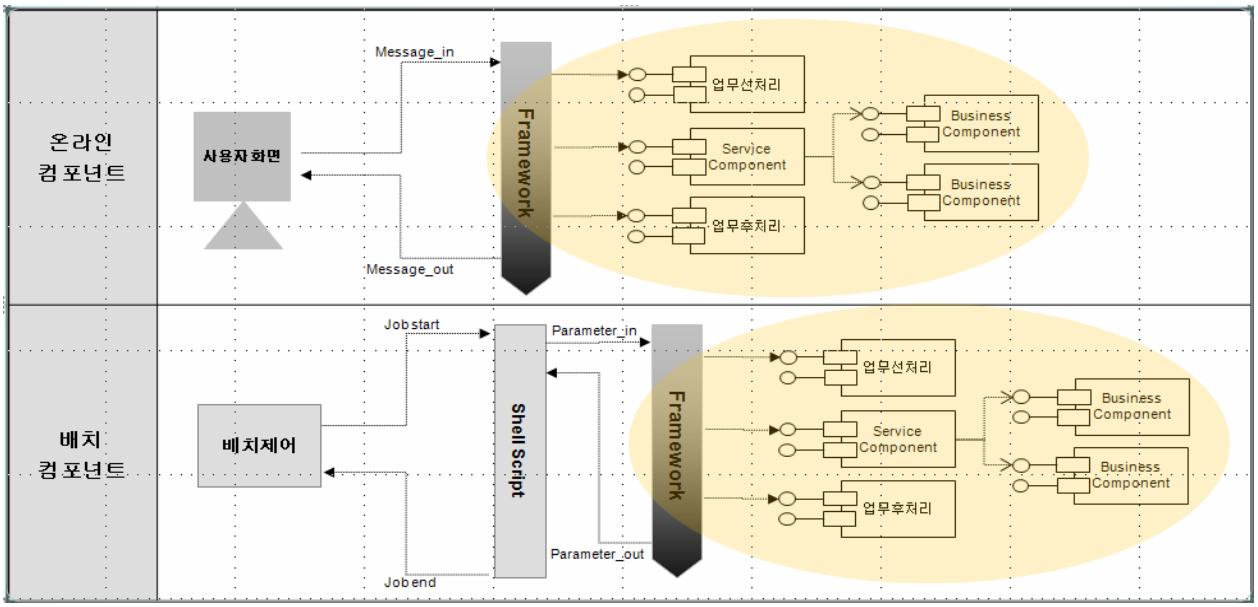
4. 금융권 차세대 프로젝트 적용 및 효과

금융권 차세대 시스템은 금융산업에서 경쟁력을 확보하고, 금융정책과 IT지원부서 간의 유기적인 조화가 가능하도록 IT인프라를 갖추는 것을 의미한다. H은행의 경우 기존의 LEGACY시스템과 시스템간의 연계를 위한 인터페이스를 포함하여 종합 연계테스트를 통해 고품질의 시스템 개발과 측정에 의한 예측 가능한 프로젝트 관리를 수행하였다.

금융권 차세대 프로젝트는 표준제정과 준수, 일관성 있는 품질정책, 계량적 관리를 통한 객관적인 프로젝트 수행관리, 지속적이고 점진적인 시스템의 개선, 변경관리의 일원화 등이 동시에 이루어지지 않으면 대규모 프로젝트를 성공적으로 완수한다는 것은 불가능하다.

4.1 로그 분석 J2EE 프레임워크

아래 [그림 4]의 H은행 시스템은 컴포넌트 및 각 클래스들이 J2EE 프레임워크에 맞게 상세 설계되어 있는 구조를 보여준다. H은행 시스템의 프레임워크 구조는 온라인과 배치처리를 하는 컴포넌트 집합으로 구분하고 있다. 그리고 각각의 컴포넌트는 서비스 컴포넌트와 비즈니스 컴포넌트로 구성되어 있는데, 비즈니스 컴포넌트는 데이터의 생성, 조회, 수정, 삭제 처리 관리 및 계산, 원장 처리 등 단위 업무처리를 담당하며 J2EE 프레임워크 패키지 계층 구조의 Business Task에 속한다.



[그림 4] H은행 J2EE 프레임워크 컴포넌트 구조

또한 서비스 컴포넌트는 서비스의 처리 흐름을 관리하고 데이터의 직접 접근을 허용하지는 않으며 J2EE 프레임워크 패키지 계층의 구조의 Service Task에 속한다.

4.2 커버리지 기법 적용 결과

본 논문의 테스트 대상은 중간에 추가 변경작업이 발생하긴 했지만 [표 3]에서처럼 최종 클래스가 592개, 메쏘드가 2,545개에 달한다.

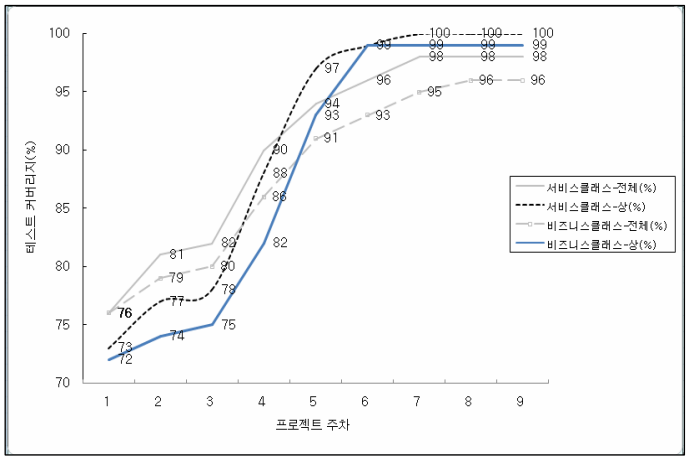
[표 3] 테스트 대상 클래스와 메쏘드 개수

업무	중요도	클래스		메쏘드	
		서비스 Task	비즈니스 Task	서비스 Task	비즈니스 Task
		전체건수	전체건수	전체건수	전체건수
AA	상	90	134	449	497
	중	108	224	726	720
	하	20	16	80	73
	소계	592		2,545	

그 테스트 대상 건수에서 알 수 있듯이 통합 테스트 단계에서 전체 대상을 커버하는 건 현실적으로 불가능하므로 중요도(상/중/하), 클래스와 메쏘드, 그리고 서비스 컴포넌트와 비즈니스 컴포넌트의 수행 클래스인 서비스 Task와 비즈니스 Task 등 최대한 세분화한 분석결과를 테스팅에 반영하는 전략으로 품질향상을 이룩하였다.

전체 테스트케이스를 매주 커버리지 분석결과에 따라 반복적으로 수행해서 오류를 최대한 발생시킨다. 앞서 설명한 커버리지 분석에 따른 테스팅 전략에 따라 중요도 '상'인 수행 클래스와 메쏘드의 서비스 Task, 비즈니스 Task 수행율이 100% 를 목표로 테스트케이스 추가 및 오류 수정을 반복적으로 수행하였다.

[그림 5] 컴포넌트 클래스 테스팅 수행율 (%)



[그림 5] 는 컴포넌트 수행 클래스의 테스팅 수행율을 표시하고 있는데 중요도가 '상' 인 서비스 Task 와 비즈니스 Task의 수행율이 각각 100%, 99% 인데, 전체는 서비스task가 98%, 비즈니스 task가 96% 이다.

금융권 대형 프로젝트 특성상 모든 클래스와 메쏘드의 테스트 커버리지를 달성하기가 어려우며 중요도 높은 것은 필수적으로 테스팅을 수행하여 오류 발견 및 수정을 반복적으로 수행하고, '상' 레벨에서 충분한 품질검증이 달성되면 '중' 레벨 및 '하' 레벨까지 범위를 확대하여 점진적인 커버리지를 넓혀가는 전략이 가장 효과적인 방법이다.

4.3 테스팅 효과

기존 테스트 관리 툴을 통해서 등록된 결함을 분석하여 커버리지 테스트를 통한 반복적 테스트 전략과 결함도출의 상관관계를 분석하였다.

[표 4]에서는 테스트케이스 건수를 증가시켜서 반복적 테스팅을 시도하면 신규 결함 발생율이 높아진다라는 것이다. 이는 결함의 발생자체가 여러 가지 원인에 기인하고 있는 상황을 파악하여야 할 필요가 있다.

[표 4] 테스트 결함 발생율

구분	결함발생 테스트 케이스 건수	수행된 테스트 케이스 건수	신규 결함 발생율	임계치 (10%)
1 주차	0	338	0.00%	10%
2 주차	0	402	0.00%	10%
3 주차	1	149	0.67%	10%
4 주차	0	285	0.00%	10%
5 주차	9	457	1.97%	10%
6 주차	1	514	0.19%	10%
7 주차	2	148	1.35%	10%
8 주차	15	504	2.98%	10%
9 주차	49	774	6.33%	10%

대부분의 SI 프로젝트는 끊임없는 요구사항의 변경에 직면한다. 이미 테스트 수행을 통해 “결함 없음”으로 선언된 프로그램이라도 요구사항에 의해 변경된 경우에는 새롭게 재 테스트 과정이 필요하며, 새로운 결함을 발생시킨다. 회귀테스트가 필요한 이유가 이러한 원인에 기인한다.

모든 비즈니스는 비용대비 효과 면에서 경제적으로 합리적인 의사결정이 될 수 있어야 하는데 본 논문에서 제시한 커버리지 기법은 중요한 온라인과 배치작업 비즈니스 컴포넌트들의 100% 테스팅이 가능하도록 객관적인 수치를 제공함으로써 테스팅 결과분석 이후에 의사결정권자가 현재 시스템 품질 정도를 파악하는데 충분한 도움을 줄 수 있다.

5. 결론

기존의 단위/통합 테스트에서는 수작업으로 수행된 결과를 개발자 혹은 테스터의 확인만을 믿고 다음 단계로 이행되는 관행이 있었다. 이는 시간이 충분하지 않은 SI 프로젝트 특성상 어쩔 수 없는 현실적 문제이다. 그러나 통합 테스트에서 결과로 도출된 시스템 로그를 분석하여 클래스 단위 혹은 메쏘드 단위로 커버리지를 분석하고 해당 시스템이 어느 정도로 테스트가 수행되었는지 판단해 본다면 수작업에 의한 테스트 문제를 어느 정도 해결할 수 있다.
또한 도출된 결함을 통해 지속적인 커버리지를 조정하여 테스트 케이스/시나리오를 진화시키고, 결함의 개연성이 높은 분야를 집중적으로 테스트하는 것이 가능하다.

향후 연구에서는 실제 시스템 오픈 이후 발생되는 결함이 과연 본 논문에서 제시한 전략에 의해 품질 개선 효과가 있었는지 여부를 확인하는 방법과 실제 적용결과를 새로운 과제로 제시한다.

테스트 기간 중에는 최대한 많은 결함 도출과 해소가 필요한 반면, 시스템 오픈 이후에는 원인 규명과 재발방지 노력이 필요하다. 충분히 품질이 입증된 시스템은 가동 이후 많은 노력과 시간을 절감해 줄 것이며, 무엇보다도 시스템 사용자와 시스템 가동 조직체의 고객에게 신뢰와 만족을 높일 것이다.

[참고문헌]

[1] Glenford Myers, "The Art of Software Testing", New York, Wiley, 1979

[2] 한동수, 정인상(2004), “ 소프트웨어 테스팅 입문”, 한국소프트웨어공학협회/ 소프트웨어 테스팅 분과

[3] Weyuker, E. J., Testing component-Based Software: A Cautionary Tale, IEEE Software, 15(5), September/October

[4] A Jefferson Offutt, Aynur Abdurazik and Roger T. Alexander, "An Analysis Tool for Coupling-based Integration Testing",Proceedings of the 6th IEEE International Conference on Engineering of Complex Computer Systems, pp 177-178, 2000