韓國SI學會誌
第3卷 第2號
2004年 11月

한국 SI산업에서 CBD사업의 장애 요인과 발전 가능성 : 한은 금융망 시스템 구축 사례

김 강 석*

The Roadblocks and Chances of Component-Based Development Business in Korea SI Industry : a Case of BOK-WIRE System

Kang Suk Kim*

Abstract

Since Component-Based Development, or CBD, struck Korea SI industry a heavy blow several years ago, many major SI companies in Korea have implemented the CBD practices in real world, leveraging supports from the government and academic world. The result, however, falls short of their expectations. The major reason is that they have mainly focused on CBD processes and technical issues around it, although it is necessary as well to deal with other non-technical issues in order to make business values from CBD. This paper begins with reminding us what CBD tries to achieve indeed in terms of software development processes, then recognizes what we have achieved so far introducing a recent CBD project case, BOK-WIRE system. Thereafter, it reveals the chances that SI companies must take in order to make their CBD efforts succeed, pointing out what are the roadblocks really.

Keyword : CBD, Software Development Process, BOK-WIRE

1. 서 론

개발 생산성과 품질은 SI조직의 입장에서 볼 때 무한경쟁의 산업환경에서 생존을 위한 가장 중요한 수단이라 할 수 있다. 이 때문에 지난 수십 년간 우리는 이를 달성하고자 무수히 많은 노력을 기울여왔다. 관리 및 개발 프로세스가 만들어 지고, 각종 도구가 만들어 지며, 개발 언어들은 그 추상성

* SK C&C R&D 센터

을 높여 좀더 쉽게 이해하고 개발할 수 있는 방향으로 발전되어 왔다. 이러한 맥락 속에서 재사용이란 개념도 오래 전부터 있어왔고 개발자 상당수가 나름의 방식대로 재사용을 하고 있다.

몇 년 전부터 컴포넌트 기반 개발(CBD)이 국내에 뒤늦게나마 상륙하여 관련 산업계에 많은 영향을 주고 있다. 객체지향이 그다지 크게 각광 받지 못한 국내의 상황을 보면 CBD의 열풍은 분명 객체지향이 주지 못한 무엇인가가 있다고 받아들여진 것이고 그 논리 중 중요한 부분을 차지하는 것이 바로 '재사용을 통한 개발 생산성 향상과 품질제고'이다(사실상 객체지향도 재사용을 지향한다). 즉, 이 열풍은 CBD를 통해 재사용을 극대화 할 수 있고, 재사용을 통해 생산성 및 품질을 향상할 수 있다는 믿음에서 출발한 것이다. 이에 따라, 산학계가 협력하여 이를 위한 개발 프로세스를 정립하고(마르미-III 등), 산출물 및 모델링 기법을 표준화 하였으며(UML), 기반 플랫폼으로서 J2EE와 .NET을 중점적으로 제시하고 있다. 이러한 성과들은 선진 프로세스 및 사례, 그리고 산업계 표준 등에 근거하여 구성되었기 때문에 나름대로의 신뢰성과 안정성을 확보하였다고 할 수 있다.

그러나 이를 현장에서 적용하기 위해서는 아직까지 많은 어려움이 남아있다. 그 중 가장 큰 문제점은 객체지향에 기반한 CBD를 하고 있는 우리가 정작 객체지향에 대한 지식과 경험이 그다지 많지 않다는 것이다. 따라서 제대로 된 컴포넌트를 도출해내고 완성도 높게 개발하는 내용적인 부분보다는 CBD프로세스와 산출물이라는 익숙지 못한 형식에 더욱 고민하고 시간을 보내는 경우가 허다하다. 이는 객체지향을 건너뛴 국내의 현실을 감안하면 당연한 결과이다. 이러한 현실 속에서 CBD가 제공할 것이라고 생각했던 생산성 향상이란 것은 상상도 할 수 없고, 재사용 가능한 수준의 자산을 만들거나 사용하는 예도 드물다. 게다가 우리는 CBD가 던지는 여러 가지 기술적인 문제들로 인해 정작 재사용의 성공에 필요한 요인들을 간과하고 있다.

우리가 여지까지 나름대로 해왔던 주먹구구식의 재사용만을 통해서는 조직의 생산성과 제품의 품질을 높일 수 없다. 우리가 지향하고자 하는 재사용은 조직적이고 체계적인(Systematic) 재사용이다. 그리고 이는 프로세스, 조직, 문화, 인력 등의 요소들이 재사용의 성공을 위해 최적화되어야만 하는 일종의 '비즈니스'이다. 따라서 CBD가 제공하는 기술적인 부분만으로는 절대로 소기의 목적을 달성할 수 없다.

본론에서는 먼저 현재 당사가 수행 중인 한국은행 금융망 시스템 재구축 사업의 내용 및 진행현황을 소개하고, 본 사업에 적용된 CBD와 연관된 기술적인 요소(프로세스, 아키텍처, 모델)의 이해를 통해 현재 한국 SI업계가 이룬 CBD의 성과를 설명한다. 이와 더불어 사업수행 과정에서 도출된 기술적 그리고 비기술적인 문제들을 제기한 후, CBD가 소기의 목적을 달성하여 개발 생산성 향상과 품질제고라는 궁극적인 목표에 기여할 수 있는 발전적인 방향에 대해 논하도록 한다.

2. 사업 진행 현황

한국은행에서 발주한 한은 금융망 시스템(BOK-WIRE) 재구축 사업은 2003년 10월부터 착수하여 2004년 9월 종료를 목표로 현재 당사가 진행 중인 사업이다. 이중 응용프로그램 개발부분에 당사의 CBD방법론(SKPE-CBD)을 적용하여 J2EE환경에서 구축되고 있으며, 본 금융망에 참가하는 금융기관들이 이를 통해 온라인 금융거래 및 내역조회업무를 웹 상에서 수행할 목적으로 추진되고 있다.

SKPE-CBD방법론이 제시하는 바와 같이 전체 개발주기를 네 가지 단계(요구획득, 아키텍처 수립, 점진적 개발, 인도)로 나누고 반복(Iteration)의 개념을 도입하여 두 번째 단계에서 아키텍처에 대한 프로토타이핑을 수행할 때 한번의 반복을, 그리고 세 번째 단계에서 두 차례의 반복을 진행하는 것으로 구성하였다.

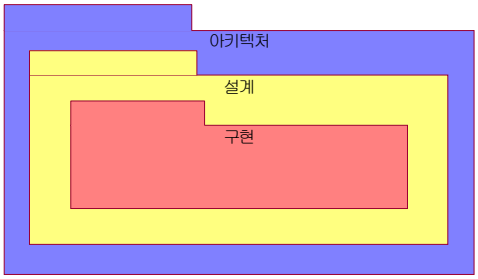
<표 1>은 개발 공정 위주로 축약하여 본 사업에 적용한 활동과 산출물을 정리한 것이다.

〈표 1〉 한은 금융망 적용 SKPE-CBD 요약

단계명	활동명	산출물명	반복
요구획득	요구사항 이해	비전기술서	
		용어집	
		요구사항기술서	
	요구사항 정의	유스케이스 모형	
		UI 프로토타입	
아키텍처 수립	요구사항 분석	유스케이스 분석모형	
		객체모형	
		UI 설계서	
	아키텍쳐 정의	소프트웨어아키텍쳐 정의서	
		매커니즘 기술서	
		컴포넌트 명세서	
		재사용 컴포넌트 명세 및 구현물	
	아키텍쳐 프로토타이핑	아키텍쳐프로토타입 설계서	1회 반복
		아키텍쳐 프로토타입	
		테스트수행 보고서	
		검토의견서	
점진적 개발	상세설계	컴포넌트 설계서	2회 반복
		데이터모형 기술서	
		데이터베이스 설계서	
		UI 상세설계서	
	시스템 구현	구현물	
		컴포넌트테스트 결과보고서	
	통합테스트	통합테스트 결과보고서	
인도	시스템 테스트	시스템테스트 결과보고서	
	인수테스트 및시스템 관찰	인수테스트 결과보고서	

3. 소프트웨어 아키텍처 수립

객체지향 혹은 컴포넌트 기반 시스템에서의 소프트웨어 아키텍처는 소프트웨어의 정적인 구조를 인터페이스를 통해 서로 연결되어 있는 서브시스템들로 정의하고, 이러한 소프트웨어 서브시스템들이 물리적인 노드 상에서 서로 상호작용하는 것을 정의하는 것이다[4]. 올바른 아키텍처를 선정하는 것은 소프트웨어 개발에 있어 매우 중요하고도 힘든 일이다. 전체 시스템의 일관성과 통합성은 바로 아키텍처를 통해서 유지되기 때문이다. 또한 복잡성을 관리하여 많은 사람이 동시에 개발을 하는 환경이 가능하게 하는 것도 아키텍처의 주요 역할이다. 다시 말하자면, [그림 1]과 같이 아키텍처라는 일종의 상호 약속된 바운더리를 미리 정의하고 그 제약조건 내에서 시스템을 설계 및 구현함으로써 일관되고 통합된 방향으로 대상 시스템을 실현한다.



[그림 1] 설계 및 구현을 제약하는 아키텍처

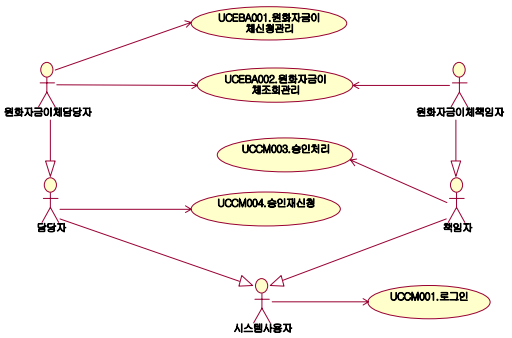
아키텍처의 중요성은 기존의 객체지향 방법론에서와 같이 CBD 방법론에서도 아키텍처 중심적인(Architecture-centric) 프로세스를 표방하고 있다는 점에서 알 수 있다. 따라서 본 프로젝트에서 어떻게 아키텍처를 수립하였는지를 살펴봄으로써 CBD의 기술적인 성과물을 확인하고자 한다.

먼저 아키텍트가 본 시스템을 둘러싼 이해당사자들의 비기능적, 즉 품질적인 요구사항을 정리하였다. 여기에는 ISO에서 정의하는 표준 품질요소를 사용하기도 하지만 본 프로젝트에서는 Grady가 정의한 FURPS+방식을 사용하였다[3]. 정리된 품질 시나리오를 토대로 아키텍처를 본격적으로 구성하는데 다음과 같이 Kruchten이 제시한 4+1 View를 적용하여 수행하였다[5].

3.1 유스케이스 뷰

요구사항 획득 단계에서 수집된 기능적인 요구사항을 바탕으로 유스케이스 모형을 작성하고 각 유스케이스 별로 상세 명세서를 작성하였다. 또한 사용자 화면 프로토타이핑을 통해 고객의 이해를

높였다. 이 과정을 통해 [그림 2]와 같이 아키텍처 수립에 가장 중요한 영향을 미치는 유스케이스를 선정하였다. 이는 시스템의 기능적인 면보다는 품질적인 요구사항에 대해 상당한 커버리지를 가지고 있는 유스케이스를 선정하여 아키텍처 수립에 활용하기 위함이다.

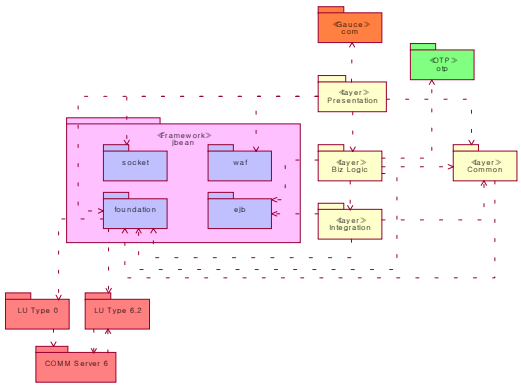


[그림 2] 아키텍처적으로 중요한 유스케이스 모형

3.2 논리적 뷰

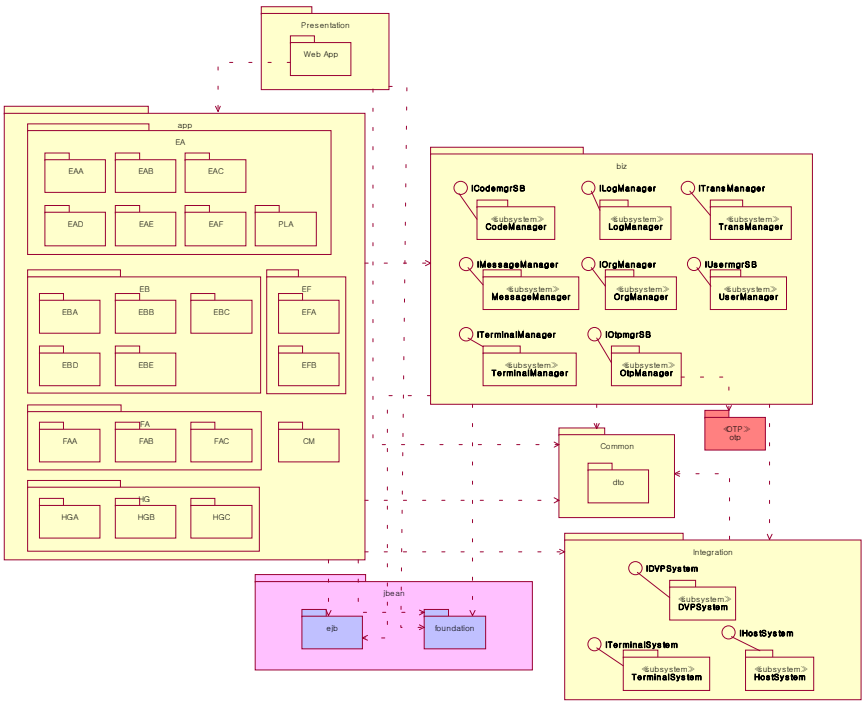
다음은 확정되어진 품질 시나리오를 기반으로 아키텍처 패턴 혹은 스타일을 결정하게 되는데, 재사용을 염두에 둔 객체지향 혹은 컴포넌트 기반 소프트웨어는 레이어드(Layered) 아키텍처를 올바르게 적용하는 것이 그 출발점이라 할 수 있다. 레이어드 아키텍처는 소프트웨어를 다수의 레이어로 구분하고 각 레이어는 좀더 일반적인 하위 레이어만을 사용하여 정의된다. 레이어는 동일한 추상화 레벨을 가진 일련의 소프트웨어로 정의된다.

레이어드 아키텍처는 매우 직관적이라 이해하기 좋고 적절한 레이어간의 추상화 및 인터페이스의 활용으로 재사용 및 하위 레이어 활용의 용이성을 확보할 수 있는 장점을 가지고 있다. 그러나, 명확하게 레이어를 분할하는 기준을 정의하기가 어렵고 또한 레이어가 많아질수록 성능이 떨어진다는 단점을 가지고 있어 신중한 결정을 요구한다. [그림 3]은 본 시스템의 최상위 수준의 논리적인 구성을 보여준다.

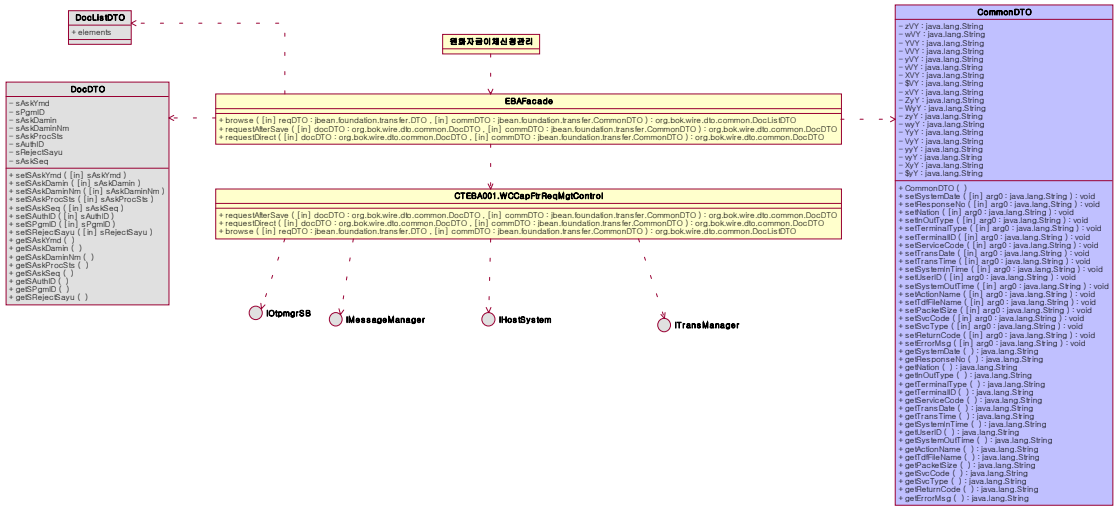


[그림 3] 논리적 뷰 - 최상위

본 시스템은 전형적인 세가지 레이어 (프레젠테이션, 비즈니스, 인테그레이션)로 분할되었고 성능을 고려하여 이들 세 레이어가 공통 레이어에 접근하는 것을 허용했다. 또한 MVC패턴을 적용하였는데, 뷰는 사용자와의 상호작용이 빈번하고 변화에 민감하므로 최 상단의 레이어에 배치하고, 사용자 요청에 따른 비즈니스 워크플로우를 담당하는 컨트롤은 프레젠테이션 레이어와 독립적으로 작동하여 비즈니스 처리의 파사드 역할을 하기 위해 비즈니스 레이어의 상단에 배치한다. 또한 모델에 해당하는 비즈니스 컴포넌트들은 도메인에 근거하기 때문에 변화 가능성이 매우 적으면서도 재사용의 가능성이 높아 비즈니스 레이어의 하단에서 콘트롤의 통제하에 놓는다. 여기서 눈 여겨 볼 것은 본 시스템이 사용할 프레임워크를 비롯한 타 컴포넌트들과의 아키텍처적인 통합을 함께 고려하고 있다는 점이다. 이것을 기반으로 각각의 레이어 별로 구성요소들간의 관계를 고려한 파티셔닝(Partitioning) 작업을 하게 되는데 아키텍처 스타일의 원칙을 벗어나지 않으면서도 파티션 내의 응집도를 높이고 파티션 간의 결합도를 낮추는 방향으로 정의하였다. 특히 비즈니스 레이어와 인테그레이션 레이어에서는 [그림 4]와 같이 각각 도메인 분석 혹은 타 시스템 연동을 통해 다양한 비즈니스 컴포넌트와 시스템 컴포넌트를 도출하여 독립적인 컴포넌트 단위로 시스템을 구성하였다.

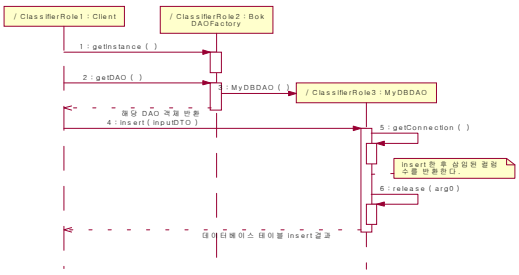


[그림 4] 논리적 뷰 - 비즈니스 레이어



[그림 5] 유스케이스 설계 - 비즈니스 구간

이렇게 도출된 아키텍처적인 주요 설계요소(클래스, 패키지, 서브시스템, 컴포넌트 등)들을 사용하여 위에서 선정한 아키텍처적으로 중요한 유스케이스들을 [그림 5]와 같이 설계하였다. 이는 설계요소간의 의존관계를 명확히 하고, 컴포넌트를 포함한 이들 요소들이 제공해야 할 서비스를 확정하기 위함이다. 그리고 이를 통해 확정된 컴포넌트의 인터페이스를 중심으로 내부설계를 수행하였다.

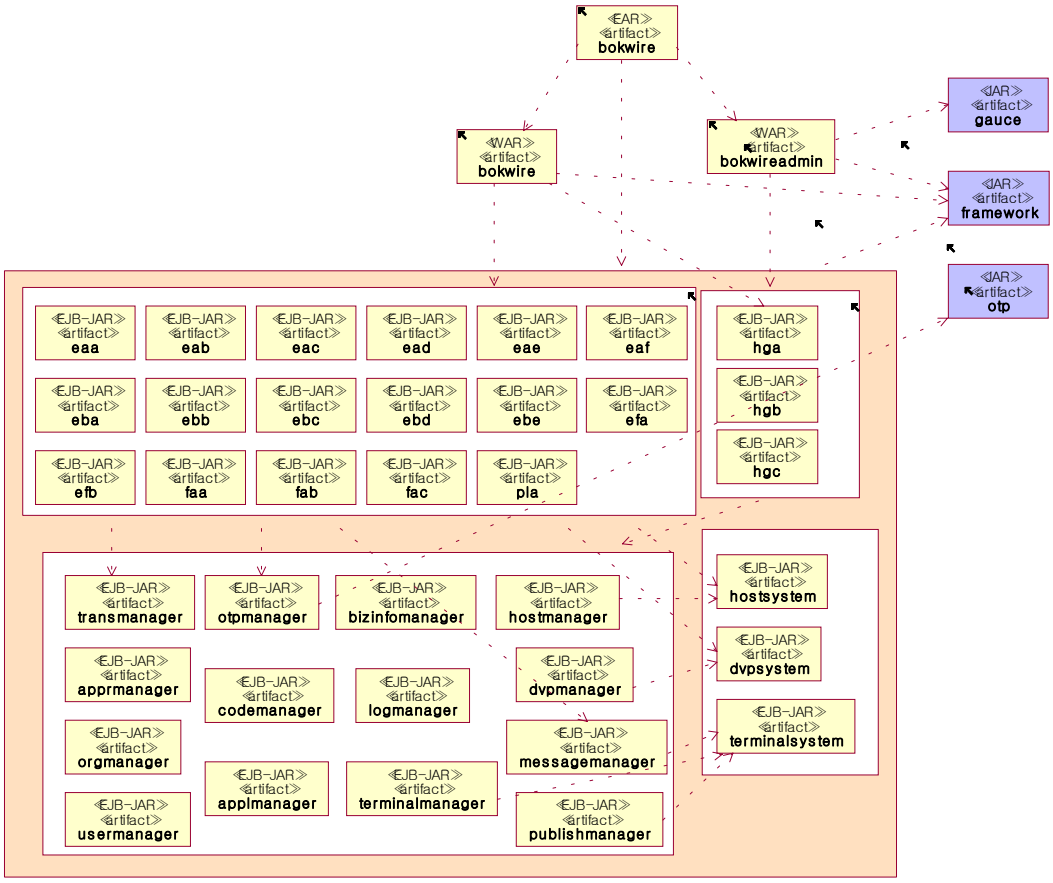


[그림 6] 저장 메커니즘 - 데이터베이스 조회

위와 같은 설계를 가능하게 하기 위해 먼저 비기능적 요구사항에서 도출하는 표준 메커니즘의 정의가 선행되어야 한다. 기능적인 요구사항이 유스케이스로 이어지는 반면, 품질 시나리오로 정리되는 비기능적 요구사항은 아키텍처 상에서 주로 하드웨어 적이거나 플랫폼 종속적인 요소로 전환된다. 이러한 기술적인 요소들과 기능중심의 분석모델이 아키텍트의 의도대로 접목이 되어야 비로서 설계모델이 도출된다. [그림 6]은 본 시스템에서 적용한 기술요소의 사례이다.

3.3 구현 뷰

논리적인 설계를 마친 다음 이를 물리적인 요소와 매칭시킨다. 파일과 디렉토리의 단위로 논리적인 설계요소들을 재정의하고 이를 패키징하여 [그림 7]과 같은 배포단위를 정의하였다.



[그림 7] 배포 정의

3.4 배치 뷰

이렇게 배포단위가 결정되면 이를 시스템 및 네트워크 구성환경을 고려하여 실제 하드웨어 노드 상에서의 최적의 배치를 고민해야 한다.

시스템 아키텍처는 초기 아키텍처를 수립할 때부터 비기능적 요구사항을 충분히 고려하여 구성한다 (사실 이 부분이 시스템의 품질에 가장 큰 영향을 미치면서 비용과 직결된다). 이 시스템 구성 위에 배포단위가 결정된 물리적인 소프트웨어를 고객의 요구사항을 만족시키는 최적의 상태로 배치한다. [그림 8]은 이러한 사례를 보여준다.

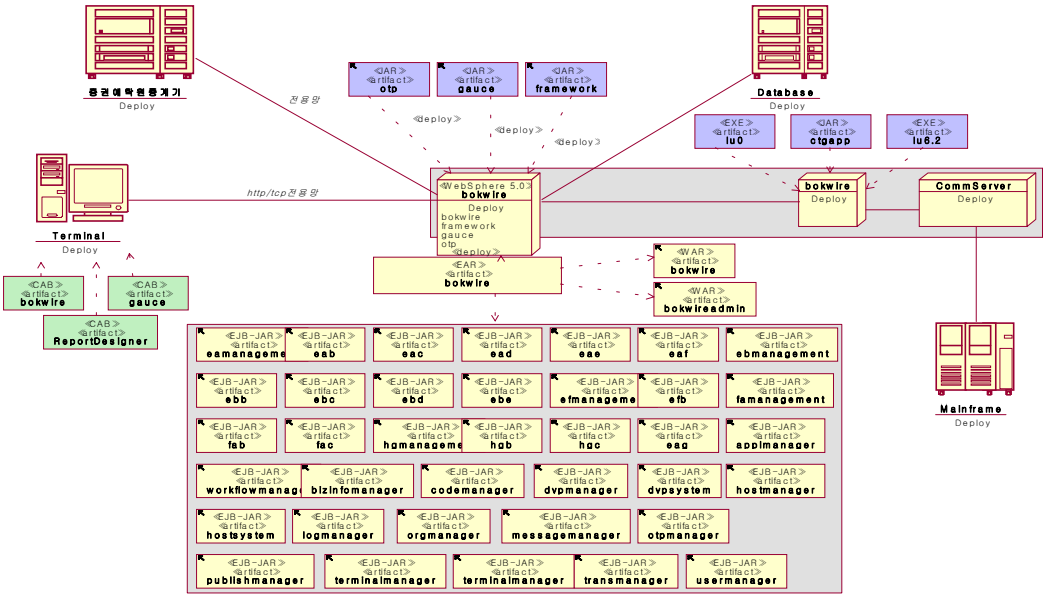
이렇듯 위에서 열거한 관점들을 중심으로 품질적인 요소에 따라 시스템의 구성을 정의하였고 이를 바탕으로 설계 및 구현을 하였다. 아키텍처에 대한 검증은 아키텍처 프로토타입을 구현하고 테스트함으로써 수행하였다.

4. CBD의 성과와 문제점

서론에서 밝힌 바와 같이 우리나라는 CBD를 도입한 지 몇 년 되지 않았지만 비교적 많은 기술적인 성과를 내고 있다. 수년 동안의 성과물이라 여겨지는 것들을 나열해보면 다음과 같다.

- 객체 지향 중심 소프트웨어 엔지니어링의 원칙(discipline) 보급
  - 반복 진화적인 프로세스의 적용
  - 아키텍처의 중요성 및 아키텍트 역할 인식
  - UML을 표준으로 한 설계 주도적 개발(Model-Driven Devlopment) 및 산출물 컴포넌트 아키텍처 수립 기법 보급
- 컴포넌트의 정의
  - 컴포넌트 도출 및 인터페이스 정의 기법
  - 컴포넌트 간 관계 설정

과거 국내에 실질적으로 도입되어 내재화되지 못했던 객체지향에 대한 원칙들이 대거 CBD의 물결과 함께 보급되기 시작하였다. UP(Unified Process)를 필두로 한 객체지향 프로세스의 영향으로 반복 진화적인 프로세스가 점차적으로 적용되기 시작하였고, 기존에 ERD나 시스템 혹은 네트워크 구성도 외에 소프트웨어 아키텍처에대한 인



[그림 8] 배치 정의

식이 부족했던 것을 새로이 체계화하였다. 표준 모델링 언어인 UML이 보급되었고 이를 활용하여 유스케이스 모델에서부터 분석 모델, 설계 모델과 구현모델, 그리고 최종적으로 테스트 모델로 이어지는 체계적인 모델링 작업과 함께 이에 대한 산출물 또한 자리를 잡기 시작했다.

이와 더불어 객체지향에서 한걸음 더 나아가 CBD를 위한 기술적인 요소로서 컴포넌트 기반 아키텍팅이 보급되었는데 컴포넌트에 대한 정의에서부터 시작하여 도메인 엔지니어링을 통한 컴포넌트 및 인터페이스 도출, 그리고 컴포넌트 간의 관계에 대해서도 실용적인 접근을 하게 되었다.

본 시스템 또한 상기한 일련의 성과를 기반으로 추진되고 있다. 앞서 살펴본 바와 같이 당사 표준의 CBD방법론에 따라 반복 진화적인 프로세스를 적용하고 있고, 프로젝트 초기에 프레임워크 기반의 잘 정의된 아키텍처를 수립하였으며, UML을 활용하여 모델 주도적인 개발을 하고 있다. 또한 도메인 엔지니어링을 통해 적절한 크기의 비즈니스 컴포넌트를 도출하고 물리적인 의존성을 낮추는 동시에 독립성을 높여 배포단위화 하였다.

그러나 SI기업의 입장에서 볼 때 CBD를 도입한 근본적인 목적인 재사용 극대화를 통한 생산성 향상과 품질제고를 달성했는가 에는 회의적인 판단을 할 수 밖에 없다. 이에 대한 문제점을 정리해보면 다음과 같다:

- CBD 적용의 기술적인 문제점
  - 개발 인력의 CBD프로세스 및 모델링에 대한 역량 부족
  - 역량 있는 소프트웨어 아키텍트의 부족
- 체계적인 재사용의 활성화 결여
  - 재사용 자산의 부족
  - 표준 준수 및 재사용 가능 컴포넌트 자산 생산 부족
  - 재사용에 대한 조직, 프로세스, 인력 등 비기술적인 재사용 확산 요인의 부재

첫 번째 문제로, 국내에서 객체지향의 도입이 뒤늦게 이루어진 영향으로 인한 아키텍트를 포함한 개발 인력의 기술적인 역량부족을 들 수 있다. 다수의 프로젝트 수행과 교육훈련을 통해 핵심인력을 육성함으로써 이 문제를 해결해야 하지만, 현실적으로 이에 대한 투자가 만만하지 않고 지식과 경험간의 괴리를 극복하지 못한 채 공정을 진행하며 많은 혼란을 겪는다. 실제로 CBD적용의 베스트 프랙티스라고 공개된 산출물 샘플들을 보면 그 내용상 미흡한 점이 적지 않게 발견된다. 또한 협력업체 의존도가 높은 SI기업의 경우에는 자체 핵심인력을 내재화할 기회조차 가지기가 어렵다.

두 번째로, 재사용 측면에서의 문제점도 거론하지 않을 수 없다. CBD프로세스를 통해 재사용 가능한 자산을 적극 활용하여 재사용 율을 높여 생산성 및 품질을 제고하고, 또한 컴포넌트화를 통해 새롭게 발굴되는 재사용 가능한 자산을 개발하여 다수의 프로젝트에서 재사용될 수 있도록 정제 및 관리하는 선 순환적인 장치역할을 하는 조직 및 프로세스가 활성화된 국내 SI기업은 드물다. 따라서 수년간의 CBD활동을 하고서도 프로젝트 단위로 단발적인 노력만 들일뿐, 전사적인 재사용 프로세스를 중심으로 이러한 노력들을 자산화하여 CBD 본연의 목적을 달성하려는 노력이 미흡하다.

첫 번째 문제점은 교육과 경험을 통한 내재화 외에는 별다른 방법이 없으므로 여기서는 두 번째 문제의 해결방안에 대해 살펴보도록 한다.

5. 체계적인 재사용

SI업계에서 CBD를 적용하는 목적을 다시 요약하면 체계적인 재사용을 통해 개발 생산성과 품질을 향상하고자 하는데 있다. [4]는 재사용이 현실적으로 어려운 이유를 설명하고 있는데 이를 정리하면 <표 2>와 같다.

이와 같은 어려움을 극복하여 재사용을 실현하기 위해서는 새로운 프로세스와 조직을 도입해야 한다. [4]는 <표 3>과 같이 재사용을 위한 네 가지의 프로세스를 소개하고 있다.

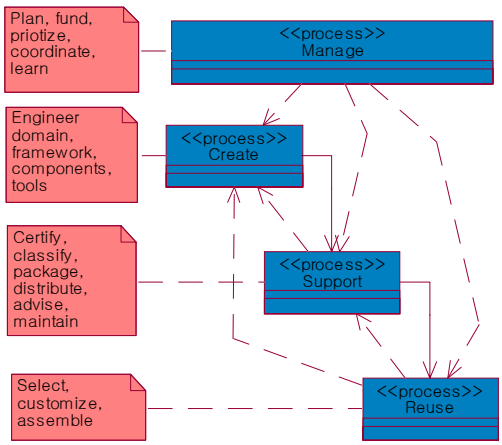
〈표 2〉 재사용을 막는 장애요인

분 류	내 용
엔지니어링	◦ 재사용 컴포넌트를 명확하게 추출 및 패키징하고, 문서화하며, 분류 저장하는 방법이 없어 재사용하기 어렵다. ◦ 컴포넌트가 유연하지 못해 대부분의 경우 재사용하지 못한다.
프로세스	◦ 재사용 컴포넌트를 만들거나 활용하는 프로세스가 없다.
조직	◦ 경영층은 재사용을 위해 장기적인 안목으로 해당 도메인 영역에 속한 일련의 프로젝트를 통해 접근해야 하나 단발적인 프로젝트 단위로만 접근한다. 해당 도메인에 속한 다수의 프로젝트를 통해 재사용 가능한 컴포넌트를 추출하고 정제해야 한다. ◦ 개발자들이 타 부서 혹은 타 개발자를 믿지 못한다. 이에 따라 재사용에 대한 막연한 회의를 가지며 사람보다는 개발 언어와 도구 같은 기술에 의존한다.
비즈니스	◦ 재사용은 투자가 필요하다. 도메인 엔지니어링을 하고, 이에 따라 재사용 컴포넌트를 만들며, 사내 컴포넌트 라이브러리를 갖추어 이를 확산하기 위해 교육 및 지원하기까지 투자가 필요하다. ◦ 재사용 자산의 개발과 활용을 통해 비용 절감과 동시에 품질 높은 시스템을 개발하고자 하는 공감대가 형성되어야 한다. ◦ 재사용에 대해 조직적인 지원과 투자를 해야 한다.

〈표 3〉 체계적인 재사용 프로세스

프로세스	설 명
Create	도메인 분석, 아키텍처 정의 및 재사용성 평가를 통해 재사용 가능한 자산을 개발(혹은 구매)하여 제공하는 프로세스
Reuse	사용자 요구사항을 토대로 도메인 분석과 재사용 자산 검토를 통해 재사용 자산을 최대한 활용하여 제품을 생산하는 프로세스
Support	재사용 자산을 검증 및 관리하고 이를 사용하도록 홍보, 분배하여 그 결과를 피드백 받는 프로세스
Manage	재사용 자산 구축의 우선순위와 일정을 수립하고, 필요한 자산의 없을 때 이의 확보를 위한 여러 대안을 검토하고 방향을 수립하는 프로세스

즉, Create 프로세스를 통해 재사용 자산을 생산해내면 Reuse프로세스를 통해 특정 시스템 개발에 활용한다. Support프로세스는 Reuse 프로세스를 지원하는 한편, Create프로세스에게 재사용 자산에 대한 피드백을 전달한다. Manage 프로세스는 상기 프로세스를 조정하고 재사용 자산에 대한 우선순위를 정하며 확보 계획을 수립 함으로써 전체 프로세스가 효과적으로 운영되도록 한다. 이를 그림으로 표현하면 [그림 9]와 같다.



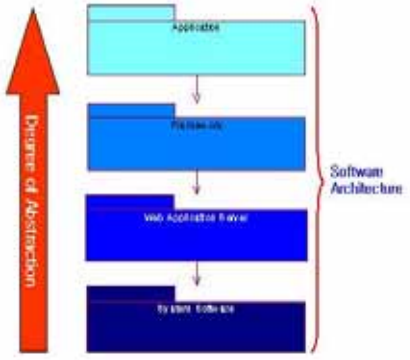
[그림 9] 체계적인 재사용 프로세스

위와 같은 프로세스를 수행하기 위해서는 Create 프로세스를 수행하는 조직(Creator)과 Reuse를 수행하는 조직(Reuser), 그리고 Support를 맡는 조직(Supporter)을 분리하여 운영해야 한다. Creator가 Reuser 조직에 속하여 특정 프로젝트를 수행한다면 시간과 예산을 최우선으로 하는 환경 속에서 자신의 목적을 달성하기 어려울 것이다. 조직이 독립적이지 못한 상태에서 프로젝트만을 통해 재사용 가능한 자산을 개발하거나 재사용을 바라기는 매우 어렵다. 그러나 그 반대로 Reuser와 Creator가 단절되어 있다면 재사용의 수요가 있는 컴포넌트를 적절한 시기에 만들어내기 불가능해진다. 따라서 양자는 항상 정보를 공유하되 서로 단기적인 목적은 달리하여 중장기적으로 생산성 향상이라는 목표를 향해 나아가도록 해야 한다. 이를 토대로 체계적인 재사용을 위해 추진되어야 할 방향을 정리해보도록 한다.

5.1 표준 프레임워크와 재사용 자산 확보

위의 사례에서 살펴 보았듯이 이러한 복잡한 아키텍팅의 과정을 프로젝트를 수행할 때마다 아무것도 없는 상태에서 매번 새롭게 시작해야 한다면 이는 매우 비효율적이다. 따라서 그 동안의 여러 가지 프로젝트의 수행을 통해 이미 조직 내에 존재하는 베스트 프랙티스, 패턴, 프로세스들을 한데 모아 이를 설명하는 모델 및 문서를 중심으로 기반 아키텍처의 표준화를 시도하는 것이 참조 아키텍처(Reference Architecture)이다[6].

이러한 참조 아키텍처가 프로젝트에 여러 번 적용되는 과정에서 잘 정의된 구현물들이 생산되면 이를 기반으로 플랫폼 기술 표준과 경험적인 베스트 프랙티스를 결합한 애플리케이션 프레임워크를 개발한다. 애플리케이션 프레임워크는 특정 애플리케이션을 구축할 수 있도록 그 기반을 제공하는 뼈대역할을 하는 것으로서 그 자체가 재사용의 대상이자 상위수준 재사용 자산을 확보하는 출발점이다. [그림 10]은 아키텍처와 프레임워크, 그리고 애플리케이션의 관계를 보여준다.



[그림 10] 아키텍처, 프레임워크, 애플리케이션

프레임워크가 제공하는 이점은 크게 두 가지로 볼 수 있다. 첫째, 프로젝트 단위 내 재사용 자산으로서의 프레임워크는 기존의 아키텍팅 과정에 투입되는 노력과 시간을 상당한 정도로 절감하면서 높은 품질의 결과물을 생산할 수 있는 기회를 제공한다. 필요한 메커니즘 중 일반적인 것은 프레임워크에서 베스트 프랙티스를 바탕으로 패턴으로 이미 구현하여 제공하기 때문에 이를 활용하면 쉽게 해결되어 고민의 시간과 노력이 크게 줄어든다. 즉, 프레임워크는 전반적인 아키텍팅의 과정에 있어서 필요한 아키텍트의 결정을 돕는다.

둘째는 프로젝트 단위가 아닌 조직 내 다수의 프로젝트에서 표준적인 프레임워크를 적용할 경우 기대되는 효과이다. 프레임워크가 조직 내의 표준으로 받아들여지고 안정화되면서 그 기반 위에서 재사용 가능한 비즈니스 컴포넌트 자산이 설계되고 구현되며 또 실제로 재사용된다. 재사용의 대상이 되는 컴포넌트를 뒷받침하고 있는 기반 아키텍처는 재사용 가능성에 있어 지대한 영향을 미치게 되는데, 이를 아키텍처 불일치(Architectural Mismatch)라 한다[2]. 이러한 문제를 가진 컴포넌트를 프로젝트 내에서 재사용하고자 한다면 아키텍처의 관점에서 고려해야 할 사항들이 늘어남에 따라 통합(Integration)에 대한 비용이 증가된다. 바로 이 때문에 현실적으로 재사용보다는 재개발을 선택하는 경우가 많다. 그러나 표준 프레임워크에서 제공하는 기반 아키텍처를 가지고 애플리케이션을 구현하게 되면 이 과정에서 생성되는 상위수준의 비즈니스 컴포넌트 자산은 해당 프레임워크를 적용한 다른 애플리케이션에서 기술적인 고민 없이 재사용될 수 있다.

5.2 조직 및 경영층의 지지와 재사용 확산

체계적인 재사용이 현실화되기 위해서는 컴포넌트나 프레임워크 같은 엔지니어링적인 요소 외에 조직적인 측면, 그리고 무엇보다도 이 모든 것이 가능하도록 할 수 있는 경영층의 지원이 필수적이다.

재사용이란 사실상 조직적인 측면에서 많은 노력과 비용을 수반하게 된다. 예를 들면, 기존의 사업수행 조직을 지칭하는 Reuser이외에 Creator와 Supporter, 그리고 Reuse Manager 역할을 수행할 개별 조직이 필요하다. 우선 중앙에서 재사용 컴포넌트 및 프레임워크를 기획하고 개발하며(설계, 구

현, 테스트), 또한 피드백을 통해 끊임없이 이를 정제하는 컴포넌트 엔지니어링(Component Engineering) 조직이 있어야 한다. 재사용 자산을 위한 레파지토리를 운영, 관리하면서 재사용을 홍보, 권장하고 실적, 문제점 등을 항상 모니터링하여 컴포넌트 기획에 참여하는 재사용 자산관리 조직이 필요하다. 또한 실제 프로젝트 현장에서 재사용 자산의 적용을 유도하고 그에 따라 교육, 가이드, 트러블슈팅을 포함한 지원을 하며 피드백을 받아 컴포넌트 엔지니어링 조직에 전달하는 컴포넌트 지원 조직이 별도로 필요하다. Reuser에 해당되는 프로젝트 수행조직은 재사용에 대한 공감대를 형성하여 단발적인 프로젝트 단위의 성과보다는 다수의 프로젝트에 걸쳐 성과를 축적하려는 노력을 기울여야 한다[4].

재사용을 현실화하기 위해 가장 우선적으로 필요한 것이 경영자의 신뢰와 전폭적인 지원이다. 이를 획득하기 위해서는 재사용의 실효성에 대해 경영자에게 확신을 심어주도록 많은 노력을 기울여야 한다. 이를 위해서 먼저 해야 할 일은 상대적으로 적은 규모로 조직 내에 구체적인 성공사례를 만드는 것이다. 여기서 추출되는 각종 긍정적인 수치자료는 경영자로 하여금 재사용 자체, 그리고 자신의 직원들이 이를 현실화할 수 있다는 자신감과 확신을 불러일으키기에 충분하다. 이러한 점진적인 접근은 여러모로 유용한데, 그 이유는 재사용이 조직에 성공적으로 정착하기 위해서는 경영층의 의지 외에 여러 가지 기술적, 조직적, 문화적 변수가 작용하기 때문이다. 따라서 점진적인 확산을 통해 수시로 추진방향을 검증 및 정제하는 것이 바람직하다.

대부분의 일들이 그러하듯이 이러한 재사용 노력의 가시적인 성과가 정량화되기 위해서는 많은 시간이 필요하다. 따라서 경영자의 입장에서는 단기적인 정량적 성과를 종용하기보다는 중장기적인 비전을 가지고 기술적, 조직적 인프라를 우선적으로 갖추어 점진적인 성장을 거치는 과정에서의 성과를 기대하는 것이 바람직하다.

5.3 재사용 비즈니스의 측정과 평가

경영자로서 중장기적인 관점으로 재사용에 대한 지속적인 투자를 하기 위해서는 재사용의 투자가치에 대한 긍정적인 평가와 이에 대한 진척도를 계량적으로 파악할 수 있어야 한다. 또한, 지속적인 측정은 재사용에 대한 노력이 제대로 진행되는지를 진단 및 관리하는 역할을 하여 조직 내 재사용의 성공적인 정착에 지대한 공헌을 한다.

정확한 측정을 위해서는 먼저 기존 프로젝트를 통해 축적된 데이터가 어느 정도 신뢰성 있는 수준으로 도출될 수 있을 정도로 탄탄한 프로젝트 관리 역량이 전제되어야 하고, 이를 활용하여 합리적이고도 측정 가능한 방법이 조직 내에서 개발되어야 한다. 이를 바탕으로 객관적이고 정량적인 목표치를 가지고 재사용이 추진되어야 하고 또한 동일한 잣대로 실제 진척도를 평가해야 할 것이다.

<표 4>는 전사적인 수준에서 재사용의 ROI를 계산하기 위해서 반드시 고려해야 하는 비용 및 이익구조와 그 사례를 잘 나타내고 있다[1].

〈표 4〉 재사용 ROI 산출 예시

측정	가정/설명	예시
(1) 개발비용	300 개발자 연간 2000시간 시간당 8만원	300 * 2000 * 8만원 = 연간 480억원
(2) 시간당 결함 수 및 결함제거 비용	1000라인 당 1결함 시간당 0.8라인	1250시간 당 1결함 1결함 당 15백만원 소요
(3) 재사용 지원을 위한 연간SW/HW 비용	Year1 : 60 user Year2 : 150 user Year3 : 240 user 전사 컴포넌트 관리자 (Year1에만 3.5억원 소요) 형상관리, 모델링 도구 비용 연간 유저당 60만원. 저장소 유지관리비 연간 사용자당 40만원	SW License: Year1 = 4.1억 Year2 = 1.5억 Year3 = 2.4억 HW/OS 비용: Year1 = 5.75억 Year2 = 0.35억 Year3 = 0.5억

(4) 재사용 지원 인력 비용	재사용 관리/지원 팀 인력 Year1 : 1 풀타임 인력 Year2 : 2, Year3 : 3	Year1 = 1.6억 Year2 = 3.2억 Year3 = 4.8억
(5) 재사용 자산 개발 비용	재사용 자산개발 비용은 일반 개발비용의 1.5배 자산 생명주기 3년 Year1 : 8400시간의 공수를 지원하기 위해 12600시간의 공수 소요 (7% code reuse for 20% of developers) Year2 : 18.5% code reuse for 50% of developers Year3 : 25% code reuse for 80% of developers	Year1 = 12600 시간 Year2 = 83250 시간 Year3 = 180000 시간 시간당 8만원이므로, Year1 = 10.08억 (3.36억*) Year2 = 66.6억 (22억*) Year3 = 144.4억 (48억*) *연간 감가상각
(6) 재사용으로 인한 직접 비용절감	재사용 비용 20% 고려	(5) * 0.8 (감가상각 포함) Year1 = 5.376억 Year2 = 35.2억 Year3 = 76.8억
(7) 결함감소로 인한 비용절감	Year1 : 8400시간의 공수를 지원하기 위해 12600시간의 공수 소요 (7% code reuse for 20% of developers) Year2 : 18.5% code reuse for 50% of developers Year3 : 25% code reuse for 80% of developers	연간 결함감소 = 재사용 사용자 비율 * 재사용율 * 480 (일반 개발시 연간 결함수) Year1 = 1.008억 Year2 = 6.6억 Year3 = 14.4억
(8) 연간 절감 비용(계)	(6) + (7)	Year1 = 6.384억 Year2 = 41.8억 Year3 = 91.2억
(9) 재사용 지원 비용(계)	(3)+(4)+(5중 재사용 자산화하기 위해 초과 투입된 .5에 해당되는 부분)	Year1 = 9.81억 Year2 = 27.25억 Year3 = 55.7억
ROI	(8)/(9) * 100	Year1 = 65% Year2 = 153% Year3 = 163%

위에서 나타난 요소 이외에도 자사의 특수성을 반영한 비용 및 이익 구조를 설정하여 이를 바탕으로 재사용 프로세스의 성과를 평가해야 한다.

6. 결 론

우리는 지금까지 한국은행 금융망 시스템 재구축 사업의 내용 및 진행 현황을 통해 한국 SI업계가 이룬 CBD와 연관된 기술적인 요소(아키텍처와 프레임워크)의 성과를 살펴보았다. 이와 더불어 사업수행 과정에서 도출된 기술적 그리고 비기술적인 문제들을 제기하였고 CBD가 소기의 목적을 달성하여 SI업계가 추구하는 개발 생산성 향상과 품질 제고라는 궁극적인 목표에 기여할 수 있는 발전적인 방향을 제시해 보았다.

소프트웨어 개발의 생산성 향상과 제품의 품질 제고라는 절대절명의 목표를 달성하기 위해서 국내 SI업계는 CBD에 많은 기대를 가지고 있다. 그러나 이를 위해서는 아키텍처와 프로세스 등으로 대변되는 기술적인 요소는 물론이고, 재사용 프로세스가 기업 내에서 정착되어 활성화되기 위한 조직, 관리, 경영층지원 등의 다양한 요소들을 함께 고려해야 한다.

기업 내에서의 재사용 확산은 단기적인 효과보다는 중장기적인 접근을 해야 하고 또한 전사적인 스케일이 필요하기 때문에 무엇보다도 재사용에 대한 경영층의 의지가 굳건해야 한다. 따라서 경영층이 자신감을 가지고 정확한 판단 하에 이를 추진할 수 있도록 재사용의 효과에 대한 객관적인 평가가 지속적이고도 전사적으로 측정되어야 한다. 이러한 평가작업이 없이 '재사용 비즈니스'를 추진한다는 것은 계기판이 고장 난 비행기를 모는 것과 같이 위험하고 무책임한 게임이 될 것이다.

참 고 문 헌

[1] Baney, G., "Keys to Successfully Calculating ROI for Enterprise Software Reuse Initiatives," *Insights*, Flashline, Sept., 2002.

[2] Garlan D., R. Allen, J. Ockerbloom, "Architectural Mismatch : Why reuse is so hard," *IEEE Software*, Nov., 1995, pp.17-26.

[3] Grady, R., *Practical Software Metrics for Project Management and Process Improvement*, Prentice-Hall, 1992.

[4] Jacobson, I. *et al.*, *Software Reuse : Architecture, Process and Organization for Business Success*, Addison-Wesley, 1998.

[5] Krutchen, P., "The 4+1 View Model of Architecture," *IEEE Software*, Vol.12, No.6 (1995), pp.42-50,

[6] Reed, P. R., "Reference Architecture : The Best of Best Practices," *The Rational Edge*, IBM Rational, Sept., 2002.

◈ 저 자 소 개 ◈

김 강 석 (kkim@skcc.com)

연세대학교 정치외교학과에서 학사, University of Arizona Eller School에서 MIS 석사학위를 취득하고, University of Minnesota Carlson School에서 박사 연구활동을 했다. 삼성SDS를 거쳐 현재 SK C&C R&D센터에 재직 중이며 전사 표준 OOAD/CBD Process, Software Architecture, Application Framework, Reusable Software Components의 개발 및 현장 적용을 담당하고 있다.