컴포넌트 기반 소프트웨어 개발 프로세스

수원여자대학 오영배
투이컨설팅 나희동
삼성SDS 박준성
고려대학교 백두권*

1. 서 론

국내에서 개발방법론이 본격적으로 적용되기 시작한 1990년대초 부터 10여년 동안 정보공학과 폭포수 개발생명주기를 기반으로한 소프트웨어 생산기술이 발전하였다. 이후 90년대 말 이후 소프트웨어 개발 프로젝트가 대형화되고 웹기반의 개방형 아키텍쳐가 확산되면서 단계적 개발생명주기와 정보공학을 기반으로한 소프트웨어 생산기술에 한계가 드러나게 되어 2000년을 지나면서 반복적 생명주기와 아키텍쳐 중심의 컴포넌트 기반 소프트웨어 공학이 대두되게 되었다.

CBD(Component-Based Development)라 함은 소프트웨어를 구축하는데 있어 부품을 조립하여 완제품을 만드는 방식이다. CBD는 한 때의 유행이 아닌 장기간에 걸쳐 발전된 소프트웨어 산업의 한 진화 형태이다. CBD는 소프트웨어 개발 기술만의 변화가 아닌 SW 산업 및 시장 전반에 걸친 구조적 변화이다. CBD 기술 자체도 앞으로 상당 기간 계속 변화가 일어날 것이며, CBD가 산업, 시장에 미치는 효과 또한 계속적인 변화가 진행될 것이다.

본 고에서는 컴포넌트 기반의 개발 기법 및 절차적인 측면의 개발 프로세스에 대하여 살펴보고자 한다. 전통적인 모델과 CBD 모델과의 비교를 통해 CBD 프로세스가 가지는 특성을 기술하고, 기존 방법론과 CBD 방법론의 차이점을 통해, 현재 나와있는 CBD 방법론들의 특징을 비교하고, CBD 기술 도입 단계를 제시하며 국내 업계의 적용사례를 통하여 CBD 개발 프로세스를 소개한다.

2. CBD 프로세스의 특징

* 종신회원

컴포넌트 기반 개발 프로세스는 전통적인 개발 프로세스와는 다른 모델을 가진다. 표 1에 전통적인 개발 방법과 컴포넌트 기반 개발 방법을 비교하였다. CBD 프로세스는 모듈화, 부품화된 아키텍쳐 중심적이며 Black Box 형태의 컴포넌트를 단위로 하고있고 점진적, 반복적, 병행적인 프로세스를 가지며 조립에 의한 개발을 지향하고 컴포넌트의 공급 및 유통을 위한 새로운 조직을 등장 시켰다[1].

표 1 개발모델의 비교

구분	전통적모델	CBD
아키텍쳐	Monolithic	Modular
컴포넌트	Implementation & White Box	Interface & Black Box
프로세스	Big Bang & Water Fall	Evolutional & Concurrent
개발방법	Scratch로부터 개발	조립
개발언어	전통언어, C, COBOL	객체지향언어, Java, C++
재사용단위	Function, ADT, Class	컴포넌트
재사용방법	Library	컴포넌트 저장소
실행플랫폼	OS	컴포넌트 플랫폼
시스템 구축기술	메인프레임/ 클라이언트 서버	웹기반 시스템
품질	개발후 테스트 중시	부품의 품질인증 및 프로세스의 품질 중시
생산성	낮음	높음
시스템특성	폐쇄적	개방화, 표준화
개발특성	제어 및 데이타 흐름 중시	Architecture centric, Interface centric
지불	Up front fee	License, subscribe
조직	Monolithic	컴포넌트 공급사, 사용사, 유통업자로 분화

2.1 아키텍쳐

CBD는 모듈 방식의 아키텍쳐를 강조한다. 따라서

시스템을 부분적으로 개발할 수 있고 컴포넌트의 추가 및 대체에 의해 기능을 점진적으로 확장할 수 있다. 이러한 설계가 가능하도록 하기 위하여 소프트웨어 시스템의 기반구조인 소프트웨어 아키텍쳐가 필요하다. 대부분의 컴포넌트 기반 시스템들은 CORBA나 IBM San Francisco와 같은 기반 소프트웨어 아키텍쳐를 따른다. 이것들은 프레임워크 형태로 제공되며 기반 소프트웨어 아키텍쳐의 참조모델로 사용된다.

프레임워크는 도메인 독립적인 것으로부터 도메인 종속적인 것에 이르기까지 계층적으로 구성된다. 표준화되고 모듈화된 소프트웨어 아키텍쳐에 의해 CBD는 비정형적(ad hoc)이고 monolithic한 설계를 피하게 된다.

특히 웹 기반의 개방형 아키텍쳐가 대두되면서 기존의 기술아키텍쳐(Techinical Architecture)와 달리 소프트웨어아키텍쳐(Software Architecture)가 중요시 되었다. 독립적으로 개발된 컴포넌트는 결국 이 소프트웨어아키텍쳐 위에서 운영되어야 하므로 CBD에서는 조직적으로 개발초기에 아키텍쳐를 구축하는 일에 집중하게 된다. 이외에 업무적인 비즈니스 컴포넌트를 구성하는 비즈니스 컴포넌트아키텍쳐까지 포함하여 최종 시스템아키텍쳐를 구성한다.

2.2 컴포넌트

컴포넌트는 소프트웨어 시스템을 구성하는 부품이며 Black Box 형태를 지향한다. 컴포넌트는 구현형태에 독립적이며 컴포넌트간에는 표준화된 인터페이스를 통하여 상호 연결된다. 컴포넌트의 형태는 제품 종속적 또는 도메인 종속적이거나 도메인에 독립적인 것들이 있다.

2.3 프로세스

CBD는 소프트웨어를 점진적으로 개발하고 인도한다. 시스템의 일부분이 컴포넌트 벤더에 의해서 획득될 수 있고, 다른 조직에 아웃소싱될 수 있으며, 다른 부분의 소프트웨어 프로세스와 동시에 진행될 수 있다.

소프트웨어 재사용을 하기 위하여, 소프트웨어 프로세스는 재사용의 관점에서 구성되어야 하며 설계자는 소프트웨어 프로세스 상에서 각각의 추상화 수준에 따른 자원을 재사용 할 수 있다. 그림 1은 전통적인 폭포수 형태의 프로세스와 CBD 프로세스의 예를 보여주고 있다[9].

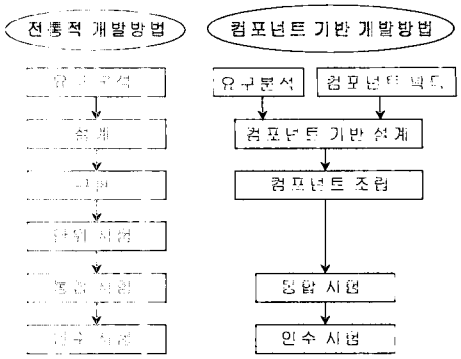


그림 1 전통적 프로세스와 CBD 프로세스

CBD 프로세스는 컴포넌트 생성과 컴포넌트 조립의 두가지 프로세스로 구성된다. 이 두 가지 프로세스는 각기 다른 조직에 의해 수행될 수도 있고, 두 프로세스가 한 조직에서 동시에 수행될 수도 있다. 전통적인 프로세스와 달리 CBD 프로세스는 컴포넌트 획득이라는 새로운 프로세스가 필요하다.

2.4 개발방법

그림 2는 CBD 개발방법의 개념을 보여준다. 그림에 보여진 것처럼 개발방법은 컴포넌트 개발과 컴포넌트 조립을 다루고 있다. 객체지향 방법과 같이 대부분의 전통적인 방법은 scratch로부터 개발이 시작됨으로써 재사용 기반의 방법에 의한 도움을 별로 받지 못한다. 구현으로부터 분리된 인터페이스를 가지고 있는 plug-and-play 소프트웨어 컴포넌트는 인터페이스를 제공한다.

CBD는 인터페이스를 통한 컴포넌트의 합성에 초점을 두고 있다. 합성은 여러 컴포넌트들의 상호 작용에 대한 설계를 필요로 한다.

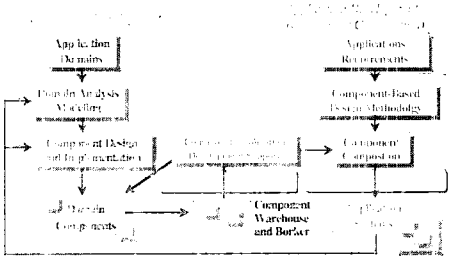


그림 2 CBD 개발방법

컴포넌트 생성과 컴포넌트 조립이라는 프로세스의 분리는 컴포넌트 유통이라는 새로운 역할을 만들었다. 컴포넌트 유통업자는 소프트웨어 컴포넌트들을 판매하고 유통할 수 있다. 컴포넌트 생성과 컴포넌트 조립이 다른 능력을 필요로 하므로 조직은 컴포넌트 벤더와 컴포넌트 사용자로 전문화된다. 이러한 전문화는 두 조직간의 중개자인 컴포넌트 유통업자를 필요로 한다.

소프트웨어 컴포넌트 벤더가 성장함에 따라 소프트웨어 컴포넌트 시장이 형성되며, 소프트웨어가 인터넷에 유통됨에 따라 웹 기반의 소프트웨어 컴포넌트 유통업이 등장하게 된다.

3. CBD 프로세스 모델의 비교

국내에서 개발방법론이 본격적으로 적용되기 시작한 1990년 초반 이후, 90년대 중반을 지나면서 국내 SI업체들은 자사의 표준 개발방법론과 사업관리 체계를 구축하고자 노력하였다. 지난 10여년 동안 이러한 노력을 통해 정보공학과 폭포수 개발생명주기를 기반으로 한 소프트웨어 생산기술이 발전하였다. 하지만 90년대 말 이후 대형 프로젝트와 금융기관 등의 대규모 소프트웨어 개발프로젝트가 많아지고 웹 기반의 개방형 아키텍처가 확산되면서 단계적 개발생명주기와 정보공학을 기반으로한 소프트웨어 생산기술에 한계가 드러나게 되었다.

이러한 배경에서 국내에서도 2000년도를 지나면

표 2 개발방법론 비교

[illegible]	[illegible]	[illegible]	[illegible]
가시성(visibility)	폭포수 모형을 기반으로 한 프로세스이므로 가시성의 확보를 위해서 지나친 문서화에 의존하게 됨. 이는 불필요한 작업요소를 증가시킬 가능성이 있으나 관리자의 이해가 쉬운 장점이 있음	반복을 통한 가시성 확보를 추구하므로 불필요한 산출물 작업을 최소화할 수 있음. 하지만 개발팀의 능력에 따라 매우 수용하기 어려운 관리난이도를 요구함	반복을 기반으로 하는 프로세스로서 가시성 확보를 위한 전략은 객체지향과 같음. 그러나 보다 체계적인 관리를 위해 컴포넌트 중심 아키텍처를 초기에 작성하여 컴포넌트를 중심으로 개발공정상의 관리 난이도를 최소화함
적응성 (adaptability)	기본적인 개발 프로세스에서 각 작업의 선택에 의한 조정이 가능함 데이타 위주의 특정 시스템 아키텍처에서만 적응성 유지가 가능함	핵심 추상개념을 중심으로 산출물 추적성을 유지하면서 조정이 가능함. 복잡한 로직을 포함한 다양한 아키텍처에서 추상개념간 추적성과 적응성을 유지할 수 있음	컴포넌트를 중심으로 기간시스템과 웹 기반 다계층 아키텍처 등 다양한 환경에서 완전한 추상개념을 유지할 수 있음. 이를 통한 적응성 확보가 유리함
이해용이성 (understandability)	단계적인 개발 프로세스 구성으로 프로세스의 이해가 용이함 그러나 분석/설계와 실제 구축 프로세스 간의 추적성 파악이 어려워 구축과정에서 각 추상개념 들의 활용성 및 이해성이 곤란함	추상개념 들이 구현 의존적이어서 추상개념 의미를 정확히 이해 하기가 쉽지 않음. 관련 전문가를 통한 명확한 반복계획이 없으면 이해가 쉽지 않음	보다 큰 단위의 컴포넌트를 중심으로 반복을 계획하여 프로세스의 이해도를 높일 수 있음 추상개념을 기존 프로세스 및 환경을 고려하여 구축할 수 있음
견고성 (robustness)	폭포수 생명주기 모형을 근거로 하여 프로젝트 진행 중에 개발사 들의 이해도 결여 등의 문제에 대처하여 프로젝트를 지속하기가 쉽지 않음	반복적 프로세스를 근간으로 하여 프로젝트 중간에 발생하는 문제점을 반영하기가 유리함. 기존 시스템 아키텍처와 같은 비객체지향 적인 추상개념으로 인한 문제를 반영할 수는 없음	다양한 환경의 수용이 가능하여 개발 중에 발생하는 문제들을 수용하기 용이함
유지보수(Maintainability)	구현위심의 추상개념이 완전하지 않아 프로세스의 변경으로 수용할 수 [illegible] 부분이 있음	기존 시스템과 정보공학 등과의 명확한 개념 연계가 쉽지 않아 프로세스 변경 요구를 수용하는데 한계가 있음	다양한 환경의 수용이 가능하며 관련 추상개념 간 관련성을 명확히 할 수 있어 프로세스의 변경 요구사항을 수용하기가 쉬움
신속성(rapidity)	설계사양서가 지나치게 추상적이어서 실제 시스템을 개발하기가 용이하지 않음	객체지향 개발 언어로 구현하기에는 완벽한 설계사양을 제공할 수 있으나, 비 객체지향 언어를 지원하기 쉽지 않음	실제 구현환경에 적합한 추상개념을 제공하여 신속한 개발이 가능한 설계사양서를 작성할 수 있음
신뢰성 (reliability)	폭포수 생명주기를 기반으로 하여 신뢰성을 유지하기 위해서는 매우 상세하고 불필요한 공정과 산출물을 작성해야 함 근본적인 위험요소 파악이 불가능함	반복을 통해 중요 위험요소를 단계별로 제거할 수 있음	반복 프로세스와 컴포넌트 기반 아키텍처를 통해 위험을 최소화하여 신뢰성 높은 프로세스를 관리할 수 있음

서 컴포넌트 기반의 반복적 개발생명주기와 아키텍처 중심 설계기술을 기반으로 한 컴포넌트 기반 소프트웨어 공학이 대두되게 되었다[3, 4, 7].

3.1 기존 방법론과의 비교

그동안 국내에서 가장 보편적으로 사용한 정보공학방법론 및 객체지향 방법론과 컴포넌트 기반 개발방법론의 소프트웨어 프로세스 측면의 특징을 비교하면 표 2와 같다.

표 2와 같이 기존 개발방법론과 CBD의 가장 큰 차이는 첫째로 반복적 개발생명주기를 적용하고 있는 점과, 둘째로 컴포넌트 단위의 재사용 체계를 제공하는 점, 마지막으로 객체지향 방법론과는 달리 기존의 다양한 개발기법을 포괄적으로 수용하고 있다는 것이다. 국내의 개발방법론과 소프트웨어 생산관리 기술의 역사가 이제 갓 10여 년 넘는 시점에서 그동안 국내 개발자들이 지나치게 절차위주의 프로세스에만 매달려 있었다면, CBD는 정보공학의 짜임새 있는 사업관리 기술과 데이타모델링 기술 그리고 객체지향 개발방법론의 어플리케이션 설계기술을 포괄적으로 수용하려는 총체적인 노력으로 이해할 수 있을 것이다[13].

3.2 CBD 방법론의 비교

CBD 주요 벤더들은 표 3과 같이 다양한 CBD 방법론을 개발하여 프로세스 관리 도구와 함께 제공하고 있다. 각각의 방법론들은 기법상의 차이점을 보이고 있으나 많은 점에서 공통점을 가지고 있다. 이들 방법론들은 신속한 반복적 개발을 위한 아키텍처 중심적이며 요구 분석 및 설계에 초점을 두고 있다. 요구사항 획득 및 관리는 효과적인 CBD 프로세스의 주요 성공요인(critical success factor)이다. CBD 프로세스는 기존의 구조적 또는 폭포수 모델에 비해 분석 및 설계에 두배이상의 노력을 들이며 코딩 및 테스팅에 반정도로 노력을 배분한다는 것이 특징이다[12].

CBD 방법론들은 또다른 공통점을 가지는데 즉, 다른 개발자 또는 개발팀들이 공통의 컴포넌트 설계 모델과 표준을 사용하도록 권장한다는 것이다. 요구분석, 공동 개발 및 사용자에 대한 반복적인 설계 검증(스토리보드, 프로토타입 등)을 이용한 요구사항 테스트 등)도 중요시하며 문서화, 컴포넌트 재사용은 도구를 사용함으로써 프로세스의 효율을 기하고 있다.

4. CBD 프로세스 도입 방안

4.1 CBD 기술 도입절차

초기 CBD를 국내에 적용하려던 업체들이 지나치게 컴포넌트의 재사용성과 UML기반 모델링 절차에만 집중하여 CBD의 실질적인 생산성 향상과 비즈니스 적시성(Time to Market) 지원에 실패한 면이 있다. 컴포넌트 기반 개발방법은 단순한 소프트웨어 프로세스와는 달리 체계적인 아키텍처와 잘 정제되고 요구변화에 강건한 컴포넌트 설계기술을 전제로 한다. 이밖에도 CBD의 궁극적인 목표인 비즈니스의 다양한 요구에 대한 정량적인 생산능력 통제가 가능하

표 3 CBD 방법론 비교

방법론	벤더	특징
SCORE	Castek	요구사항을 정의하기 위해 프로젝트 초기에 반복적 프로토타이핑을 강조 Sterling의 CoolGen과 같이 사용
Catalysis	Icon Computing	패턴을 프로세스에 도입하여 패턴별 적용지침에 따라 다양한 형태의 프로젝트에 유연하게 적용하도록 함
UML Client Server Object Oriented Process	James Martin	아키텍처, 데이타 중심 모델링 및 가능적 분할을 강조 James Martin사의 Consulting 및 SI의 강력한 지원을 받음
Progression	MTW	레거시 시스템의 기능 및 인터페이스 명세를 잘 식별하고, 지연되는 개발에 대한 프로젝트 역할과 접근방법을 지원함
OPEN	OO Process Environment & Notation Consortium	계약(contract) 명세, 팀단위 분석 및 검사에 대한 과업 및 기법에 특징이 있음
Complete Life Cycle Incremental Process(CLIPP)	CA/Platinum	use case 모델링, 상세명세 개발, 컴포넌트 테스트, 컴포넌트 분류 및 분산 객체 개발에 대한 아키텍처의 지원방법을 제공
Rational Unified Process(RUP)	Rational Software	어플리케이션 중심적이고 비즈니스 중심적인 미들웨어와 시스템 소프트웨어 계층에 대한 use case와 컴포넌트 아키텍처를 강조
Team Fusion	HP	JAD를 강조하며 점증적 개발 및 위험 분석에 가중치를 둠

기 위해서는 비즈니스 모델링과 이의 관리기술 확보가 선행되어야 한다. 이러한 CBI의 특징은 일반 업체에서 컴포넌트 기반 소프트웨어 생산기술 확보를 어렵게 하는 주된 요인이 되고 있다.

국내업체들이 개발방법론과 객체지향 설계기술 및 비즈니스 모델링 기술들을 습득하여 컴포넌트 기반 개발기술을 획득하는 과정에는 다음과 같은 여섯 단계의 과정을 거치게 된다. 본 모델은 국내환경에서 국내업체들이 컴포넌트 기술을 습득하여 실질적인 개발생산성과 품질의 향상을 달성하기까지 거치게 되는 단계를 컴포넌트 개발기술 성숙도 모형(Component Development Technology Maturity Model : CDMM)으로 정리하여 CBI기술을 도입하려는 국내 업체들이 참조토록 하기 위함이다. 기본적인 CDMM 각 단계에서 달성할 목표와 수준은 표 4와 같다[5].

표 4 컴포넌트 개발기술 성숙도 단계(CDMM)

5단계 (Organization)	✓ 발생가능 한 비즈니스 요구에 대한 정량적인 생산능력 통제(Product Line) ✓ 요구 조정에 따른 아키텍처의 최적화(동적 컴포넌트 영향 평가) ✓ 신규 어플리케이션 아키텍처 예측 생산
4단계 (Business)	✓ 재사용 컴포넌트 리파지토리의 구축 ✓ 정량적인 아키텍처 평가 및 컴포넌트 적응성, 안정성 통제 ✓ 비즈니스의 요구에 대한 아키텍처 영향도 평가 및 반영
3단계 (Architecture)	✓ 팀간 기술적 협업이 가능한 수준의 컴포넌트 분할 ✓ 프로젝트 네트워킹을 통한 지식의 재사용 ✓ 체계적인 아키텍처 관리 (Framework & Architecture Style적용)
2단계 (Component)	✓ 반복 프로세스를 통한 핵심 요구관리를 수행함. ✓ 비 정형적 아키텍처를 구현할 수 있음. ✓ 패턴기술을 적용한 체계적 설계 기술 습득이 주 목표임.
1단계 (Process)	✓ 산발적인 기술의 적용으로 결과를 보장하지 못함. ✓ 명확한 아키텍처가 없으며, 컴포넌트를 식별하지 못함. ✓ 모델링 기법 및 개발 프로세스 습득이 주 목표임.
0단계	

첫 번째 단계에서는 통합모델링언어(UML)와 비교적 정형화 된 개발프로세스를 습득하는 것을 주 목표로 한다. 이는 일반적으로 컴포넌트 기술을 프로세스(Process) 측면, 제품설계기술(Product)측면, 업무기술(Business)측면으로 나눌 때 가장 먼저 CBI로 이행하려는 조직에서 습득해야 하는 기술이 프로세스 기술임을 반영한 것이다. 2단계를 거치면서 반복적 개발프로세스에 익숙하게 된 조직은 별도의 핵심 제품설계팀을 구축하게 되고 이 팀을 중심으로 패턴과 아키텍처 스타일과 같은 제품설계기술을 습득해야 한다. 3단계는 제품설계기술이 안정화되어 조직 수준의 프레임웍과 아키텍처가 구축된 상태이며 4단계부터는 이들 아키텍처와 비즈니스와의 관련성 평가 및 통제가 가능한 단계이다. 이 단계에서는 아키텍처 평가기법(Evaluation Software Architecture)과 비즈니스 모델의 리파지토리 구축이 주된 목표이며 최종적으로 5단계에서는 제품계열(Product Line) 조직이 완성되어 체계적으로 고객의 요구를 소프트웨어 제품의 품질과 생산능력에 반영할 수 있게된다[8].

컴포넌트 기술은 향후 소프트웨어 산업을 산업화로 전환하게 하는 기초기술임에 틀림없다. 따라서 그 동안 다양한 측면에서 연구해 온 소프트웨어 생산기술을 총체적인 접근이 필요하며 컴포넌트 기반 개발방법론이 이에 해당한다. 이러한 이유로 국내 업체들은 보다 포괄적인 차원에서 CBI를 접근해야 하며 CDMM은 이를 위한 참조모형이 되기를 바란다.

4.2 국내 적용사례

일반적으로 컴포넌트 기반 방법론을 적용하려는 대상 조직은 팀웍과 관련 기술의 숙련을 위해 상당기간 경험을 필요로 한다. 아울러 대규모 프로젝트에 적용하는 경우에는 프로젝트 초기에 안정적인 개발환경 및 개발조직의 구축을 위해 상응하는 노력을 기울여야 한다. 그러나 국내 업체들의 프로세스 성숙도 수준은 평균 CMM 레벨 1.5 수준으로 파악된다. 따라서 보편적인 국내 개발자들의 수준을 고려하여 CBI를 적용하려는 조직은 안정적인 개발생산성 확보를 위해 다음과 같은 부분을 감안하여야 한다.

- 프로젝트 초기에 개발환경 및 아키텍처 이해를 위한 충분한 교육 및 파일럿 프로젝트가 필요함.
- 각 단계간 반복 과정에서 프로젝트 중반 이후의 공정상 유연성 인정 필요. 프로젝트 초기와는 달리 개발조직의 성숙도가 증가함에 따라 산출물 및 개발공정을 단축하여 생산성을 향상할 수 있음.
- 각 단계별 핵심 산출물을 통한 개발관리의 필요. 프로젝트 개발환경의 안정화와 함께 중반 이후에 불필요한 공정이나 산출물로 생산성을

저하하지 않도록 각 단계별 평가기준에 맞는 핵심 산출물만을 공정평가 대상으로 하여야 함.

- 개발 초기에 형상관리 및 개발 리파지토리 구축이 필요함. 핵심 산출물을 각 반복을 통해 정련하는 과정에서 반드시 버전관리 및 변경관리가 필요함.

국내 환경에서 그 동안의 단계적 개발생명주기에 익숙한 개발자들이 CBD의 반복적 개발생명주기를 완전하게 적용하는데는 현실적인 한계가 있음을 인정해야 할 것으로 본다. 이러한 제약하에서 반복적 개발생명주기를 적용함이 사용자 요구사항과 프로젝트 개발범위를 프로젝트 기간 중에 수시로 변동할 수 있음을 의미하는 것이 아님을 이해하는 것이 필요할 것이다. 또한 개발자들은 반복적 개발생명주기를 적용하는 것이 기존의 단계적 개발생명주기보다 훨씬 더 명확한 단계구분과 목표설정이 필요함을 이해해야 할 것이다. 이러한 상황에서 다음은 마르미III[2]를 토대로 실제 국내 모은행에서 적용하고 있는 CBD개발 프로세스를 소개한다. 이는 반복적 개발생명주기 적용의 현실적인 한계와 정보공학의 장점인 데이타 중심 모델링 기법 및 객체기술을 적절히 조합한 사례이다.

개발방법론은 우선 계획단계, 아키텍처 단계, 점진적 개발단계, 인도단계의 네 단계로 구분된다. 그리고 각 단계는 명확한 목표설정과 이의 달성을 위한 실행 그리고 평가를 포함하는 작은 프로젝트 수준으로 정의한다. 이는 국내 개발자들의 문화가 단계적 개발생명주기에 익숙한 점을 감안하여 프로젝트 관리 및 계약 등은 단계적인 접근으로 수행하되 내부적으로는 반복적 접근 방법을 인정한 것이다.

각 단계의 구분 기준은 다음과 같다.

- 계획단계 : 개발 대상 시스템의 명확한 범위를 확정하며, 이를 검증하기 위한 UI수준의 구현물을 개발함.
- 아키텍처 정의 단계 : 시스템 구현을 위한 모든 아키텍처적인 위험요소가 제거되어 최종 시스템 아키텍처를 확정하고 각 컴포넌트 도출을 완료함.
- 개발단계 : 모든 컴포넌트 들이 개발되고 통합테스트까지 마무리 하였으며, 아키텍처 정의단계에서 정의된 테스트 기준을 통과함.
- 인도단계 : 실제 사용자가 사용하도록 교육과 설치작업을 실시하기 위한 모든 작업이 완료되었으며, 시스템테스트와 인수테스트 기준을 관족시킴.

각 단계는 프로젝트 일정 수립과 진척관리 과정에서 주요관리이정표(Major Milestone)로 활용하게 되며, 국내 개발자들의 문화를 고려하여 각 주요관리이정표는 계약 및 기성고 산정과 연계하여 관리하게 된다. 그 외 각 단계에는 몇 개의 활동이 있고 각 활동은 활동의 목표를 달성하기 위하여 일련의 작업을 정의하여 실시한다. 각 활동은 프로젝트 관리 과정에서 보조관리이정표(Minor Milestone)로 활용된다. 이 외에 개발단계에서 몇 개의 미니프로젝트를 도출하고 각 미니프로젝트는 점진적(Incremental), 반복적(Iterative)으로 개발 대상 컴포넌트를 분할하여 시스템을 개발하게 된다. 미니프로젝트는 보조관리이정표로 관리하게 된다.

이 외에 각 산출물은 상세한 작성지침과 사례를 포함하여 제시하게 되며, 각 주요관리이정표 시작단계에서 개발자 교육을 실시한다. 또한 아키텍처 단계와 개발단계를 거치면서 비즈니스 컴포넌트의 실행환경이 되는 기술 아키텍처와 소프트웨어 아키텍처를 먼저 설계, 구현하고 추후에 비즈니스 컴포넌트의 설계방식을 결정하고 상세 비즈니스 컴포넌트를 구현한다.

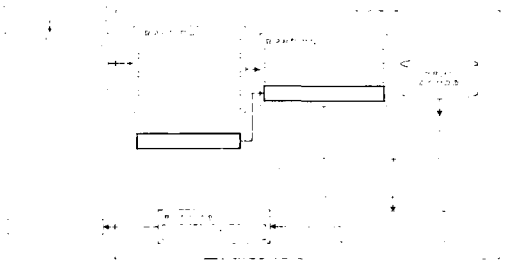


그림 3 계획단계 작업흐름도

다음은 각 단계에 대한 설명과 작업흐름도 이다.

계획단계(그림 3)에서는 개발 할 시스템의 비전, 목표 및 범위를 결정하는 작업을 수행한다. 이를 위해 사용자 요구사항을 수집하고 이를 비즈니스 유즈케이스 모형과 활동도를 통한 업무모형을 작성하고 시스템 유즈케이스 및 업무객체 모형을 작성한다. 이를 통해 사용자 요구사항과 시스템 개발범위를 최종 확정하고 전체 프로젝트 규모 및 소요비용을 추정한다. 또한 시스템 개발을 위한 프로젝트 계획을 확정하고 품질보증계획을 수립한다.

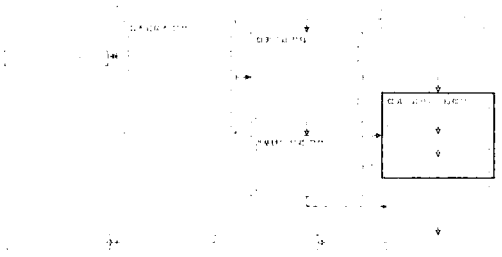


그림 4 아키텍쳐 정의 단계 작업흐름도

아키텍쳐 단계(그림 4)에서는 요구사항 정련을 통해 아키텍쳐 관련 요건을 파악하고, 이를 토대로 구현 가능한 응용 아키텍쳐와 재사용성 높은 비즈니스 아키텍쳐를 포함하는 시스템의 아키텍쳐를 결정한다[8]. 또한 다양한 기법을 통해 비즈니스 컴포넌트를 식별하고, 응용 아키텍쳐를 지원하는 응용 컴포넌트를 결정한 후 최종 컴포넌트 명세를 작성하여 컴포넌트를 도출한다[10].

아울러 아키텍쳐 프로토타입을 통해 아키텍쳐적인 위험요소를 발견하고, 이를 반영하는 시스템 아키텍쳐를 결정하여 향후 구현 과정에서 발생 가능한 아키텍쳐 위험요소를 제거한다. 그 외에 점진적으로 시스템을 개발하기 위한 계획을 수립하여 다음 단계를 위한 반복계획을 수립한다. 이 단계에서는 반드시 독립적으로 실행 가능한 기능을 구현하여 사용자가 시스템 구현에 적극적으로 참여하여 요구사항을 제시하려는 분위기를 유도해야만 실질적인 구현상의 위험요인을 완화할 수 있게 된다.

비즈니스 컴포넌트를 도출하는 가장 효율적인 방법은 유즈케이스를 중심으로 비즈니스 컴포넌트의 크기를 결정하고 지속성 계층(Persistency Layer), 즉 데이타 관점의 클래스를 중심으로 인터페이스를 식별하는 것이 적절하다.

점진적 개발단계(그림 5)는 개발 해야 할 시스템을 유스케이스를 기준으로 미니프로젝트 단위로 분할하여 개발한다. 미니프로젝트별로 구축한 유스케이스를 구현 관점에서 보완하고, 이를 바탕으로 아키텍쳐를 또한 구현 관점에서 보완한다. 식별된 컴포넌트의 내부를 설계하고, 데이타베이스, 사용자 인터페이스 등을 설계하고 구축한다. 개발된 컴포넌트에 대해 인터페이스가 제대로 구현되었는지 검사하기 위해 단위테스트를 수행하고, 컴포넌트간 통합 테스트를 수행하여 실제 가동환경에서 통합하게 된다. 각 미니프로젝트가 끝난 후, 개발된 시스템의 기능성 및 성능이 사용자의 요구를 만족하는지 확인하기 위해 시스템테스트를 실시한다.

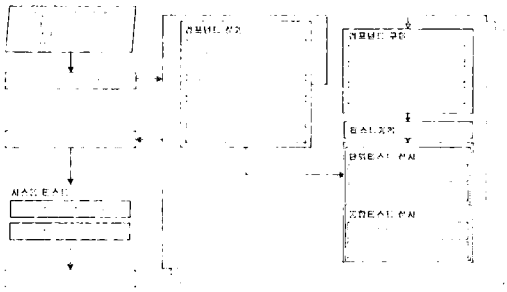


그림 5 점진적개발단계 작업흐름도

인도단계(그림 6)에서는 개발자 환경에서 개발된 결과물을 컴포넌트 리포지토리에, 또는 실제 시스템이 운영될 사용자 환경에 설치한다. 기존에 운영되고 있는 리포지토리나 시스템이 있을 경우 신규 시스템으로 전환하여 원활한 운영이 가능하도록 한다. 개발된 컴포넌트 또는 시스템에 대하여 최종적으로 사용자 요구사항과의 일치 여부에 대하여 승인을 얻고 프로젝트의 모든 전달물을 사용자에게 전달하고 인도하게 된다.

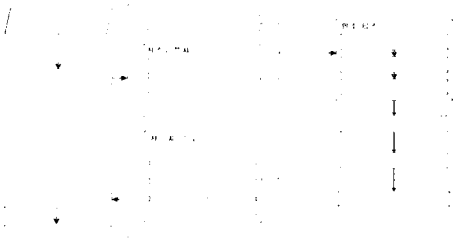


그림 6 인도단계 작업흐름도

5. 결 론

우리 나라 소프트웨어 산업이 소프트웨어 공학의 기본원칙을 적용하는데 있어 미흡하다는 것은 부인할 수 없는 현실이다[6]. 국내 소프트웨어 기업들은 그동안 제조(Programming) 능력 확보에 많은 투자를 해왔으나 최근 들어 SI산업의 규모가 팽창함에 따라 우리 기업들은 제조 능력보다는 설계(Architecting) 능력의 확보가 필요함을 인식하고 있고 개발방법론 및 프로젝트 관리 기술의 필요성을 절실히 느끼고 있다.

CBD는 아키텍쳐 중심의 소프트웨어 구축 방법이며 코딩보다는 요구 분석 및 설계에 노력을 많이 들이고, 반복적, 병행적 프로세스를 지향하고 있다. 최근들어 우리 나라는 금융권을 중심으로한 여러 기업들이 CBD 프로세스 도입을 준비중이거나 추진 중에 있다. CBD 프로세스를 성공적으로 도입하기 위해서는 장기적인 관점에서 전사적인 단계적 도입전략을 수립하여야 하며 기업 내에 축적되는 지식, 제도 및 문화로 구축되는 장기적인 핵심 역량으로서의 소프트웨어 프로세스 및 아키텍쳐를 구축해 나가야 할 것이다.

참고문헌

[1] 오영배, 박준성, "CBD 적용사례 연구", 한국정보과학회 소프트웨어공학회지 제12권 제3호, 1999.9.

[2] ETRI, 마르미 III 작업지침서, 2001.9.

[3] 전자신문, "CBD, 새로운 IT개발방법론으로 각광", 2001.10.

[4] 한국전자통신연구원, "컴포넌트 기술 산업동향 자료", 2001.10.

[5] 나희동, "Architecture driven Component Development" : SoftEXPO 2001 발표자료, 2001.11.

[6] 박준성, "CBD 개론", 한국소프트웨어컴포넌트컨소시엄, 2001.12.

[7] 한국소프트웨어컴포넌트컨소시엄, '알기쉬운 컴포넌트' 2002.2.

[8] Bass, L.; Clements, P.; & Kazman, R. Software Architecture in Practice. Reading, MA: Addison Wesley, 1998.

[9] Aoyama, M. "New Age of Software Development", Proceedings of ICSE Workshop on CBSE, Kyoto, Japan, Apr 1998.

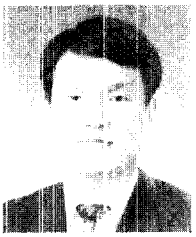
[10] Kazman, R.; Barbacci, M.; Klein, M.; Carriere, S.J.; & Woods, S.J."Experience with Performing Architecture Tradeoff Analysis."Proceedings of ICSE99. Los Angeles, CA, May 1999.

[11] Clements, P. & Northrop, L. "A Framework for Product Line Prac tice."Version 2.0, July 1999.

[12] Light, M. "Component Development : Getting Past Projects," Gartner Group, July 1999.

[13] Linden, A. "2001 Hype Cycle of Emerging Trends and Technologies," Gartner Group, July 2001.

오 영 배



1982 고려대학교 기계공학과 졸업(학사)
1985 인하대학교 전자계산학과(공학석사)
현재 고려대학교 컴퓨터학과 박사과정
1988 정보처리기술사
1995 ~ 1997 미국캘리포니아대학(UCI) 객원연구원
1983 ~ 2000 한국전자통신연구원 책임연구원 신상
2000 ~ 현재 수원여대 컴퓨터응용학부 교수
관심분야 컴포넌트공학(CBSE), 소프트웨어재사용기술, 소프트웨어품질관리, 전사상거래
E-mail:yboh@suwon.c.ac.kr

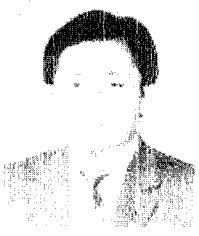
나 희 동

1983 전남대학교 산업공학과 졸업(공학사)
1988 정보처리기술사 정보관리 부문 취득
2000 서울산업대학원 전자기산학과 석사(공학석사)
2000 Carnegie Mellon University MSE 수료
1983 ~ 1995 이스턴컨설팅(주) 개발팀장
1996 ~ 1999 한더정보시스템(주) IT솔루션 사업부
2000 ~ 현재 투이컨설팅(주) 수석 컨설턴트
관심분야Software Architecture, CBSE, Software Process, DW/DM, ESP/ISP
E-mail:hdna@2c.co.kr

박 준 성

1978 서울대학교 경영학과 졸업(학사)
1983 서울대학교 경영학과(석사)
1988 미국 Ohio 주립대 전산학 및 시스템공학(박사)
1987 ~ 1988 미국 Louisiana 주립대 조교수
1988 ~ 2000 미국 University of Iowa 부교수
1995 ~ 2000 KAIST, 포항공대, 중국 청화대학, 미국 MIT 등 초청교수
2001 ~ 현재 삼성SDS 상무, CTO 및 CKO, 첨단소프트웨어공학센터장, 정보기술인 대상
관심분야 컴포넌트공학(CBSE), 데이타베이스, 정보통신, 소프트웨어공학, OR
E-mail:jpark@samsung.co.kr

백 두 권



1974 고려대학교 수학과 졸업(학사)
1977 고려대학교 산업공학과(공학석사)
1983 미국 Wayne State Univ. 전산과학(석사)
1986 미국 Wayne State Univ. 전산과학(박사)
1986 ~ 현재 고려대학교 컴퓨터학과 교수
정보통신대학장
관심분야 데이타공학, 메타데이타, 컴포넌트시스템, 데이타베이스
E-mail:baik@swsys2.korea.ac.kr