□특집□

소프트웨어 기술의 변혁과 발전

양 해 술[†] 이 하 용[††] 황 석 형[†††]

◈ 목 차 ◈

1. 서 론
2. 소프트웨어의 변혁
3. 새로운 시대의 소프트웨어
4. 소프트웨어 개발 기술의 패러다임 변화
5. 새로운 소프트웨어 개발의 비전
6. 소프트웨어 신시대

1. 서 론

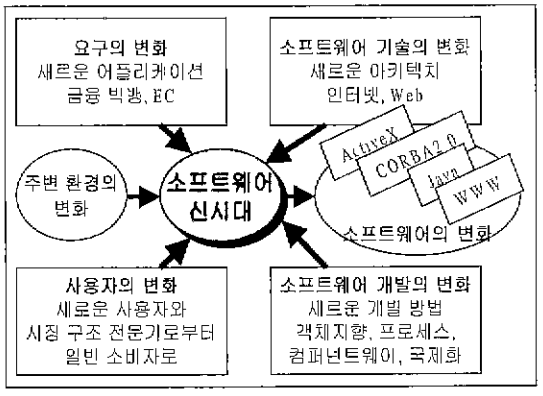
소프트웨어의 새로운 시대가 시작되고 있다. 개인용 컴퓨터와 인터넷 및 인트라넷의 급속한 발전으로 빅뱅이라 부를 만큼 비즈니스 구조의 변혁이 있었고, 기술과 비즈니스의 내면으로부터 소프트웨어의 변혁이 추진되고 있다. 동시에 최근 10년간에 연구 개발된 새로운 소프트웨어 기술이 실용 단계를 맞이하고 소프트웨어 개발 현장도 여러 가지 기술의 급격한 흐름 속에 신시대를 맞이하고 있다.

밀레니엄 시대의 소프트웨어 개발 기술은 다양한 관점에서 변혁과 발전을 지속할 것으로 전망된다. 먼저, 사용자 요구의 변화로 인한 새로운 어플리케이션이 지속적으로 개발될 것이며, 인터넷과 Web 환경의 등장으로 네트웍 지향 소프트웨어 아키텍처가 주축을 이루는 환경으로 변화해 갈 것으로 예상된다. 또한, 소프트웨어의 사용자가 전문가로부터 일반 소비자로 변화해 감으로써 엔드 유저 컴퓨팅(EUC)이나 엔드 유저 개발(EUD) 환경으로 이동될 전망이며 객체지향, 컴퍼넌트웨어 등 소프트웨어 개발 기술에 대한 패러다임에도 큰 변화가 예상된다.

그러나 이 새로운 소프트웨어 기술은 무엇을 목표로 하고 어디로 가고 있는 것인가? 이 변혁을 겪고 새로운 소프트웨어 기술을 활용하고, 새로운 비즈니스를 창조하는 일을 어떻게 해야 할 것인가? 본 고에서는 밀레니엄 시대를 맞이하여 기술과 비즈니스의 양면에서 소프트웨어 신시대의 비전을 기술해 보았다.

2. 소프트웨어의 변혁

먼저, 소프트웨어 변혁의 방향을 살펴보면 (그림 1)과 같이 4가지 방향의 변화로써 그 흐름을 파악할 수 있다.



(그림 1) 소프트웨어 변화의 흐름

† 종신회원 호서대학교 벤처전문대학원 교수
†† 종신회원 : 한국소프트웨어품질연구소 선임연구원
††† 종신회원 : 선문대학교 정보공학부 조교수

2.1 요구의 변화 : 소프트웨어 빅뱅

소프트웨어가 비즈니스 활동을 단순히 지원하던 것에서 비즈니스 활동 자체가 되거나 전략적 의사결정을 하는데 필요한 요소가 되고 있다. 금융 빅뱅이나 전자상거래, 비즈니스 활동의 변혁은 그것을 실현하는 새로운 소프트웨어가 계속적으로 필요하고 그 변화가 지속적으로 필요하기 때문에 소프트웨어의 빅뱅이 요구되고 있다.

2.2 사용자의 변화 : 소프트웨어의 시민혁명

PC의 폭발적인 보급에 따라 소프트웨어는 이미 일부 전문가를 대상으로 하는 것이 아니고, 다수의 일반 소비자를 대상으로 하는 소비재가 되고 있다. 즉, 소프트웨어 대중화의 시대에 살고 있다. 따라서 종래 전문가 집단에서 암묵적으로 알고 있는 지식은 통용되지 않는다. 또한, 일반 사용자는 디자인이나 가격에 민감하며 하드웨어의 급속한 가격 저하에 부응하는 소프트웨어의 생산성 향상이 요구되고 있다.

2.3 소프트웨어의 변화 : 네트워 지향 소프트웨어 아키텍처

Web으로 상징되는 인터넷 기술의 급속한 발전으로 인해 새로운 소프트웨어 아키텍처가 등장하고 있다. Web을 중심으로 하는 사용자 인터페이스, 분산 객체 환경과 컴퍼넌트웨어 등 네트웍을 중심으로 다양한 어플리케이션을 제휴할 수 있는 소프트웨어 아키텍처가 요구되고 있다.

2.4 개발의 변화 : 새로운 개발 기술과 개발 모델

객체지향 등의 새로운 개발 기술과 더불어 소프트웨어 개발 조직의 분산화 및 국제화는 이미 피할 수 없는 추세가 되고 있다. 특히, 1990년대의 비즈니스는 스피드가 중시되고 있으므로 소프트웨어 개발에서도 생산성과 품질 향상 및 개발 기간의 단축이 요구되고 있다. 또한, 종래의 소프트웨어 개발 기술은 소프트웨어를 변화하지 않는 것으로 전제하고 있었으나 이제 변화를 전제로 하는 소프트웨어 개발 기술이 요구되고 있다.

3. 새로운 시대의 소프트웨어

새로운 시대의 소프트웨어란 어떤 것인가? 여기에서는 기술면에서 몇 가지 특징을 추출해 보기로 한다.

3.1 새로운 어플리케이션의 창조

인터넷의 보급은 그룹웨어, 전자상거래, CALS (Continuous Acquisition and Life-Cycle Support)/EDI 등의 네트웍을 활용하는 새로운 유형의 어플리케이션을 창조하였다. 이러한 어플리케이션에는 다음과 같은 특징이 있다.

- 네트웍을 이용
- 소프트웨어 중심의 어플리케이션
- 콘텐츠와 유통

등과 같이 종래의 어플리케이션 개념으로는 이해할 수 없는 새로운 개념의 어플리케이션이나 서비스를 포함한다.

3.2 새로운 사용자의 출현

소프트웨어의 대중화로 인해 지금까지의 소프트웨어의 형태, 특히 GUI(Graphical User Interface)가 아직 일반 사용자에게는 사용에 어려움이 있다는 점이 새롭게 부각되고 있다. 이로 인해 종래의 GUI 기본 구조를 다시 숙고한 새로운 GUI가 등장하고 있다. 예컨대, Gentner 등에 의한 HotJava Views에서는 풀다운 메뉴바나 오버랩핑 윈도우즈, 더블 클릭 등 사용성의 관점에서 편리하지 않은 인터페이스가 삭제되고 있다. 이것은 편리하지 않은 기능을 제거했다는 점에서 C++에 대한 Java의 개발 사상과 비슷하다. 향후, 컴퓨터와 가전 제품

이 융합되면서 사용자 인터페이스의 문제는 한층 더 중요해 질 것이다.

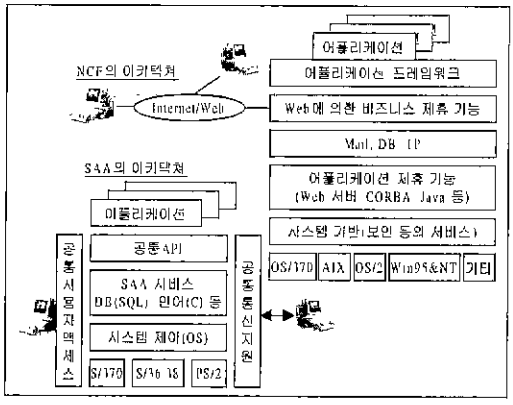
더욱이, 다양한 사무용 어플리케이션 패키지가 제공되고 소위 엔드유저 스스로가 이와 같은 패키지를 조합하여 보다 나은 이용이 가능한 엔드유저 컴퓨팅(EUC) 또는 엔드 유저 개발(EUD)이 보편화될 것이다.

한편, VRML(Virtual Reality Modeling Language)이나 3D 그래픽스 기술의 발전은 프로그래밍이라는 개념에도 영향을 주고 있다. 예컨대, 비디오 게임 스타일의 애니메이션을 프로그램 언어로 이용하는 방법이 제안되고 있다. 또한, 최근의 RPG(Role Playing Game)는 극히 정교하고 사용자 자신이 스토리를 작성할 수 있는 게임도 발매되고 있다. 이것은 게임이 프로그램 언어의 역할을 하고 있으므로 말하자면 궁극적인 EUC/EUD인 것이다.

3.3 새로운 소프트웨어 아키텍처와 개발 기술

Web은 집중 또는 일체로 된 소프트웨어 아키텍처로부터 네트워을 중심으로 하여 모듈화로 분산된 소프트웨어 아키텍처로 전환을 촉진시켰다.

약 10여년 전, 메인 프레임 개발 업체들로부터 시스템 아키텍처가 발표되었으나 10년이 지난 지금, 분산 객체나 Web을 중심으로 하는 새로운 아키텍처가 발표되고 있다. 예를 들면, (그림 2)는



(그림 2) SAA와 NCF

IBM에 의한 SAA(System Application Architecture)와 NCF(Network Computing Framework)를 대비한 것이다.

소프트웨어 아키텍처의 변화는 다음과 같은 의미가 있다.

3.3.1 복잡한 계통을 전제한 아키텍처와 개발 기술

현재, 복잡한 사회 활동을 반영한 소프트웨어를 단일 구조로 실현하는 것은 곤란하다. 여러 가지 서브 시스템을 조합시킨, 예를 들면 복잡한 계통을 전제로 조합하는 것이다.

지금까지의 소프트웨어에서는 요구 기능의 증대에 따라 대규모 프로그램 코드의 개발이 필요하다고 생각할 수 있다. 그리고 단일 구조였기 때문에 프로그램 내부의 복잡도는 규모가 커지는 것 이상으로 증가하게 된다. 또한, 종래의 소프트웨어 개발 기술은 대규모의 복잡한 프로그램 코드를 개발하는 것이 목적이었다. 그것은 벽돌을 쌓아 고층 빌딩을 짓는 것과 같다고 비유할 수 있다. 그러나 컴퍼넌트웨어나 Java의 출현으로 소프트웨어 아키텍처의 모듈화나 플랫폼에 따르지 않는 소프트웨어의 실행이 가능해지고 있다. 이제 소프트웨어는 하나씩 작성하는 시대로부터 조합시키는 시대로 변화하고 있다. 즉, 소프트웨어의 다운사이징이 요구되는 시대이다.

3.3.2 네트워을 전제로 하는 아키텍처

Web의 보급으로 인해 네트워을 전제로 한 소프트웨어 아키텍처가 요구되고 있다. 예를 들면, 사용자는 Web 페이지 등의 네트워 상에서 분산된 자원을 통일성 있게 보급할 필요가 생겼다. 이 때문에 사용자 인터페이스도 GUI에서 NUI(Network User Interface)로 진화하고 있다. 예를 들면, Windows95가 GUI에 대응하고 Windows98은 NUI를 목표로 한 것이라고 할 수 있다.

3.3.3 변화를 전제한 아키텍처와 개발 기술

소프트웨어는 업무용이든 개인용이든 사용 형태의 변화에 대응하여 진화하고 성장하지 않으면 안된다. 필자의 감리 경험으로도 대규모 소프트웨어의 경우 개발 비용의 70~80%가 기능 추가에 소요되는 경우도 있다. 이것은 기능이나 인터페이스의 변경 등에 대응하여 소프트웨어가 변화를 계속하는 것을 전제로 하지 않으면 안된다는 것을 의미한다.

그러나 종래의 소프트웨어 아키텍처와 그 개발 방법은 변화를 전제로 고려하지 않았다. 오히려 소프트웨어의 기능 변경을 보수라고 부르고, 버그 수정 등의 번거로운 문제로 생각하였으나 이것은 소프트웨어의 본질을 파악하지 못한 부적절한 지적이다.

4. 소프트웨어 개발 기술의 패러다임 변화

변화에 대응할 수 있는 소프트웨어 개발 기술은 어떠한 모습일까? 소프트웨어 개발을 개발 대상인 프로덕트와 개발 방법인 프로세스의 2가지 측면으로 보고, 새로운 소프트웨어 개발 기술을 살펴보기로 한다.

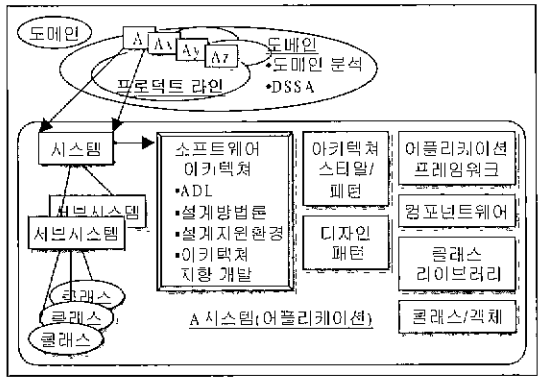
먼저, 프로덕트의 기술로서 소프트웨어 아키텍처와 컴퍼넌트웨어에 초점을 맞추어 설명하고, 프로세스의 기술로서 우수한 소프트웨어 개발 프로세스를 소개하기로 한다.

4.1 소프트웨어 아키텍처

변화에 대응할 수 있는 소프트웨어를 구현하기 위해서는 소프트웨어 전체의 포괄적인 구조의 형태를 묻지 않을 수 없다. 이 문제는 소프트웨어 아키텍처로서 수년간 연구 개발이 진전되어 왔다.

종래에도 소프트웨어 아키텍처의 사고 방식이 있었지만 소프트웨어의 각 계층에서 아키텍처를 설계하거나 재사용하는 구조는 없었다.

이것에 대해 (그림 3)과 같이 소프트웨어 아키텍처를 기술하는 아키텍처 기술 언어 ADL(Architecture Description Language), 아키텍처 설계 방법론, 아키텍처 설계 지원환경, 아키텍처 스타일(패턴), 아키텍처 지향 개발(Architecture-Based Development) 등 광범위한 연구 개발이 진행되고 있다.

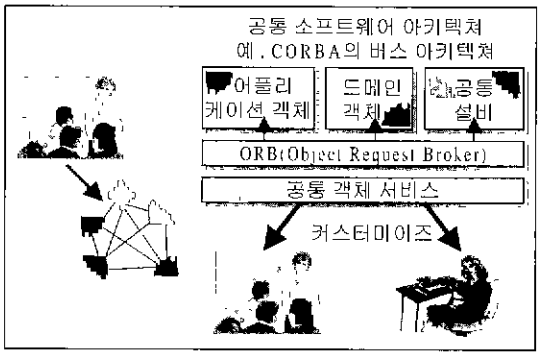


(그림 3) 아키텍처의 기술

동일한 업무나 어플리케이션 영역의 소프트웨어 아키텍처는 유사하다고 생각할 수 있다. 이와 같은 유사한 어플리케이션의 모음을 어플리케이션 도메인(Application Domain)이라 부른다. 동일한 어플리케이션 도메인 내에서는 어플리케이션의 아키텍처는 유사하고, 도메인 고유 소프트웨어 아키텍처 DSSA(Domain-Specific Software Architecture)라 부른다. 그 중에서 동일 제품 계열의 어플리케이션군을 프로덕트 라인(Product Line)이라 부른다. 실제로 아키텍처의 설계는 고도의 기술을 요구하는 것으로 단독으로 설계하는 것이 곤란하며 시간도 많이 소요된다. 현장에서도 과거의 유사 아키텍처를 본보기로 하여 개발한 것이 많다. DSSA나 프로덕트 라인은 아키텍처 전체의 재사용을 목표로 하고 있다.

객체지향에서는 아키텍처에 따라 클래스를 구조화한 것을 어플리케이션 프레임워크라 부르며 이것은 어플리케이션의 한 형태라고 볼 수 있다.

방법론의 측면에서는 사용자의 요구로부터 소프트웨어 아키텍처를 개별적으로 설계하는 것이 아니고 공통의 아키텍처를 기본으로 어플리케이션을 개발하는 방법론이 제안되고 있다. (그림 4)는 이 두가지 아키텍처 설계의 차이를 CORBA를 예를 들어 표현하였다.

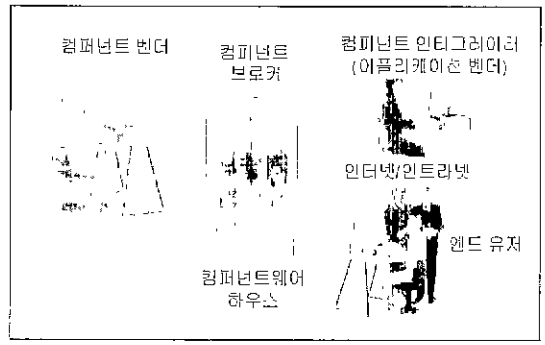


(그림 4) 공통 아키텍처에 의한 설계

종래의 개발 방법론은 특정한 아키텍처 도메인에 의존하지 않는 범용 방법론에 주안점을 두고 있었다. 그러나 범용 방법론은 실제로 적용할 때 구체성이 부족하고 효과에도 한계가 있다. 앞으로는 도메인 분석·설계 등의 기술과 결합된 도메인이나 아키텍처를 전제로 한 개발 방법론이 요구되고 있다.

4.2 컴퍼넌트웨어

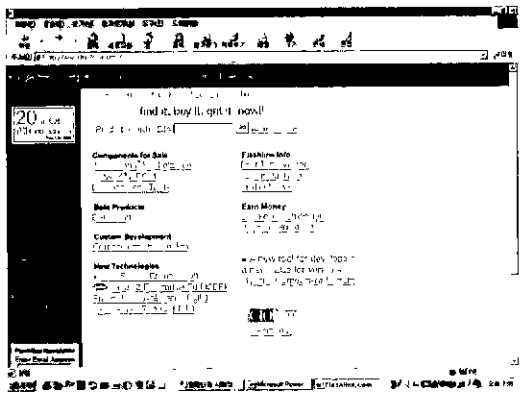
컴퍼넌트웨어란 조합하고 컴파일하여 그대로 이용할 수 있는 Plug & Play형 소프트웨어 부품을 조합시켜 어플리케이션을 구축하는 기술이다. 초기의 컴퍼넌트웨어는 Windows를 주체로 하는 말하자면 독립형(standalone) 컴퍼넌트웨어였다. 지금은 네트웍 상에서 부품을 제휴하는 분산 컴퍼넌트웨어로 진화하고 있다. 대표적인 컴퍼넌트웨어 아키텍처로서 ActiveX/WindowsDNA, CORBA (Common Object Request Broker Architecture), JavaBeans가 있다. 이것들은 분산처리 환경에서 부품을 조합시키는 기반이 되는 분산 객체 환경을 일체로 하여 제공한다. 국내에서도 업무 어플리케이션 개발로 컴퍼넌트웨어를 적용한 예가 나타나고 있다. 따라서 특정 업종이나 업무 분야의 컴퍼넌트 표준화도 진행되고 있다. 이와 같은 하드웨어나 소프트웨어의 범용 부품을 COTS(Commercial Off-The-Shelf)라 부른다.



(그림 5) 소프트웨어 개발의 재구조화

이와 같이 컴퍼넌트웨어 기술의 발전은 소프트웨어 개발 형태, 또는 소프트웨어 산업 구조의 재구조화를 재촉하고 있다. 그 징조는 국내외에서 소프트웨어 개발이 부품 개발과 부품을 조합시켜 어플리케이션을 개발하는 컴퍼넌트 인티그레이터(integrator)로 발전된다는 점이다. 국내에서도 소프트웨어 컴퍼넌트의 벤더(vender)가 나타나고 있으며 이를 기반으로 정부에서도 컴퍼넌트를 구매하기 위한 정책을 추진하고 있다.

따라서, 이와 같은 재구조화는 필연적으로 소프트웨어 부품의 유통업(컴퍼넌트 브로커)을 발생시킨다. 특히, 소프트웨어는 네트웍에서 유통될 수 있기 때문에 네트웍 상에서 소프트웨어의 판매, 유통이 미국을 중심으로 시작되고 있다. (그림 6)은 소프트웨어 부품 유통업을 하는 인터넷 사이트의 예이며, 이러한 유통업의 등장으로 소프트웨어 부품의 라이센싱 등 새로운 문제도 제기되고 있다.

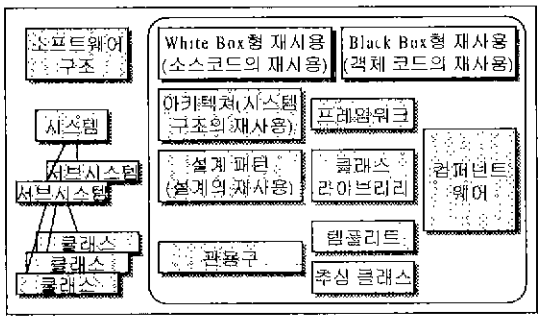


(그림 6) 인터넷 상에서의 컴퍼넌트 브로커의 예

4.3 객체지향 재사용 기술의 진화

객체지향이 목표로 하는 것 중 하나는 재사용이다. 초기에는 계승에 의한 코드의 재사용, 또는 차분(差分) 프로그래밍이 이용되었다. 이것을 구현의 계승이라 부른다. 그러나, 구현의 계승은 밀접한 계승 관계이고 상위 클래스의 구현 변경이 하위 클래스에 영향을 미치는 '연약한 기본 클래스 문제'가 생긴다. 이것에 대응하여 인터페이스의 재사용이 착안되고 인터페이스의 형태를 계승하는 인터페이스 계승(interface inheritance)이 사용되고 있다.

한편, 1980년대 말부터 (그림 7)에 나타낸 것처럼 재사용은 시스템의 계층에 따라 그 정밀도나 이용 방법이 달라진다고 생각되고 있다. 이들의 재사용 부품은 다음과 같이 분류할 수 있다.



(그림 7) 각 계층에서의 재사용

4.3.1 설계의 재사용과 프로그램의 재사용

좋은 설계를 재사용하는 구조로서 시스템의 아키텍처를 재사용하는 소프트웨어 아키텍처나 국소적인 객체군의 설계를 재사용하는 디자인 패턴이 제안되었다. 소프트웨어 아키텍처는 포석, 패턴은 정석에 대응된다.

아키텍처에 부합하여 객체를 구조화하고 말하자면, 어플리케이션의 반제품을 어플리케이션 프레임워크라 부른다. 패턴이나 프레임워크는 커스터마이즈하여 사용한다. 커스터마이즈를 용이하게 하기 위해 변경개소를 미리 명시해 두는 방법이 사용되고 있다. 이와 같은 개소를 hot spot 또는 확장점(extension point)이라 부른다. 최초의 프레임워크라고 일컬어지는 MVC(Model-View-Controller)는 지금은 GUI의 기본 프레임워크로서 널리 이용되고 있다.

4.3.2 범용 부품과 도메인 고유 부품

부품은 처음에는 도메인에 의존하지 않는 범용 부품이 주를 이루었다. 그러나 부품의 이용 효과를 높이기 위해서는 특정 도메인 고유의 요구나 도메인 고유 소프트웨어 아키텍처에 적합한 부품이 필요해진다. 특히, 비즈니스 어플리케이션 쪽의 부품은 비즈니스 객체라고 불리고 있다. 도메인 고유 부품의 업계 표준화도 제안되고 있다. 예를 들면, 유통업에서는 POS용 부품의 표준화가 이루어지고, 이것을 이용한 POS가 제품화되고 있다.

4.3.3 분산 객체 환경

클라이언트/서버 어플리케이션을 개발하기 위해서는 클라이언트와 서버 간에 객체를 이용할 수 있는 구조가 필요하다. 이것을 지원하는 기반 환경으로서 CORBA(Common Object Request Broker Architecture)나 DCOM(Distributed Component Object Model) 등의 분산 객체 환경이 제공되고 있다.

4.4 소프트웨어 개발 프로세스

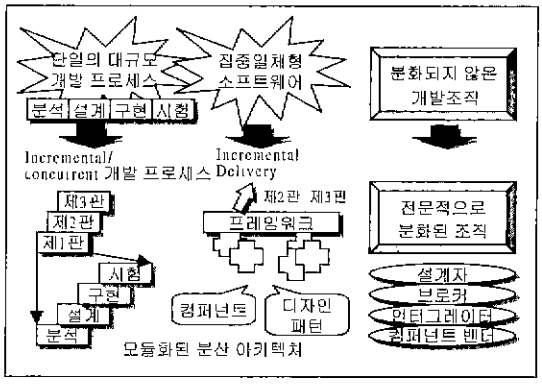
개발 프로세스는 개발의 절차를 누구든지 이해할 수 있도록 형식적으로 모델화하고 개발자 각자의 배후에 있는 개발 방식의 구조적 문제를 해결하는 것이다. 1987년의 프로세스 프로그래밍의 제안에 의해 소프트웨어 개발 프로세스의 연구와 개발이 본격화되었다.

지금까지의 개발 프로세스는 단일 릴리이즈(release)를 가진 폭포수형 모델이 중심이었다. 이 개발 프로세스는 대규모의 복잡한 소프트웨어 개발에 적용되고 있었다. 그러나 지금 새로운 소프트웨어 개발에 요구되고 있는 것은 개발 기간의 단축과 변화(성장, 진화)에 신속히 대응하는 것이다. 이로 인해 단계적으로 소프트웨어를 개발하고 제공하는 Incremental Delivery가 주목되고 있다.

특히, 객체지향 개발의 도입이나 RAD(Rapid Application Development)의 도입에서는 Incremental한 개발 프로세스에 의한 Incremental Delivery가 적용되고 있다.

따라서 개발 프로세스의 개선을 위한 우수한 방법으로서 ISO9000 시리즈 인증이나 CMM(Capability Maturity Model) 또는 SPICE를 이용한 프로세스 심사와 개선이 주목되고 있다.

5. 새로운 소프트웨어 개발의 비전



(그림 8) 소프트웨어 개발 형태의 전환

새로운 소프트웨어 개발의 모습으로 현재 개발하고 있는 우수한 소프트웨어 개발의 비전을 종래의 소프트웨어 개발과 비교하여 (그림 8)에 도시하였다.

소프트웨어 아키텍처, 개발 프로세스, 개발 조직을 일체화된 구조에서 모듈화된 구조로 전환하여 유연하고 우수한 소프트웨어를 개발하는 것을 목표로 한다.

6. 소프트웨어 신시대

소프트웨어 개발의 여러 분야에서 새로운 패러다임이 도래하고 있다. 이 점에서는 종래의 전문가를 중심으로 하는 닫힌 소프트웨어 구조로부터 사용자와 사회에 열린 소프트웨어 구조가 필요해지고 있다.

한편, 이 분야에서는 미국 기업을 중심으로 치열한 개발 경쟁이 전개되고 있다. 이와 같은 국제적인 현황을 살펴볼 때, 우리 나라 정보산업 분야에서도 이와 같은 기술의 변혁기에 대비하여 소프트웨어 연구 개발에 대한 대응 방안이 체계적으로 모색되어야 한다.

변혁기에는 기회도 있는 법이므로 우리 나라의 소프트웨어 기술자와 연구자가 새로운 기술에 도전하여 소프트웨어 개발의 신시대를 열었으면 하는 바램이다.

참고문헌

[1] Aoyama, M, "Agile Software Process and its Experience", Proc. ICSE'98, Apr. 1988.

[2] Booch, G., "Object Solutions - Managing the Object-Oriented Project", Addison-Wesley, 1995.

[3] Gentner, D. and Nielson, J., "The Anti-Mac Interface", CACM, Vol. 39, No. 8, pp. 70-82, Aug. 1996.

[4] Selker, T., "New Paradigms for Computing", CACM, Vol. 39, No. 8, pp. 28-69, Aug. 1996.

[5] Shaw, M, and Garlan, D., "software Architecture", Prentice Hall, 1996.

[6] Mowbray, T. J. and Ruh, W. A., "Inside CORBA", Addison Wesley, 1997.

[7] Cockburn, A., "Surviving Object-Oriented Projects -A Manager's Guide", Addison Wesley, 1998.

[8] 深澤良彰, 本位田眞一(編), "特集「オブジエクト指向分析・設計」" 情報處理, Vol. 35, No. 5, May 1994.

[9] 本位田眞一, 青山幹雄, 深澤良彰, 中谷多哉子(編著), "オブジエクト指向分析・設計-開發現場に見る實踐の秘訣", 共立出版, 1995.

양 해 술

1975년 홍익대학교 전기공학과 졸업(학사)
1878년 성균관대학교 정보처리학과 졸업(석사)
1991년 日本 오사카대학 정보공학과 소프트웨어공학 전공(공학박사)
1975년-1979년 육군중앙경리단 전자계산실 시스템분석장교
1986년-1987년 日本 오사카대학 객원연구원
1980년-1995년 강원대학교 전자계산학과 교수
1993년-1994년 한국정보과학회 학회지편집부위원장
1994년-1995년 한국정보처리학회 총무이사, 논문편집위원장
1994년-1998년 한국산업표준원(KISI) 이사
1995년-현재 한국소프트웨어품질연구소 소장
1998년-현재 한국정보처리학회 S/W공학연구회 위원장
2000년-현재 호서대학교 벤처전문대학원 교수
관심분야 : 소프트웨어공학(특히, S/W 품질보증과 품질평가, 품질감리와 인증, 품질컨설팅, OOA/OOD/OOP, CASE, SI), 소프트웨어 프로젝트관리

황 석 형

1991년 8월 강원대학교 전자계산학과 조기 졸업(이학사)
1992년 日本 오사카대학 기초공학부 연구생
1994년 日本 오사카대학 대학원 정보공학과(공학석사)
1997년 日本 오사카대학 대학원 정보공학과(공학박사)
1997년-현재 선문대학교 정보과학부 조교수
관심분야 : 소프트웨어공학(특히, Adaptive Object-Oriented Software, Formal Method), Adaptive Programming 기법, CAI 등

이 하 용

1993년 강원대학교 전자계산학과 졸업(이학사)
1995년 강원대학교 대학원 전자계산학과 소프트웨어공학 전공(이학석사)
1996년-1997년 경희대학교 공과대학 전자계산공학과 강사
1995년-현재 한국소프트웨어품질연구소 선임연구원
1997년-현재 경원대학교 공과대학 전자계산학과 강사
관심분야 : 소프트웨어공학(특히, S/W 품질보증과 품질평가, 품질감리, 객체지향 프로그래밍, 객체지향 분석과 설계, CASE)