소프트웨어 공학

千 柳 植
(한국전자통신연구소 행정전산망주전산기개발본부 본부장)

■ 차 례 ■

[1] 개 요

소프트웨어 공학은 사용성(usability), 신뢰성(reliablity), 유지보수성(maintainablity) 등을 갖춘 소프트웨어를 경제적으로 획득하기 위하여 공학적인 원칙을 확립하고 사용하는 기술이라고 정의할 수 있다. 소프트웨어 공학의 대상은 소프트웨어 자체보다 소프트웨어 산물(엄밀하게 말하면 소프트웨어 시스템의 산물)로 하는 것이 더 실감이 난다. 소프트웨어 공학의 제1차적인 목적은 소프트웨어 산물(software product)의 품질 향상과 생산성 향상에 있다. 소프트웨어 공학은 크게 기술적인 분야와 관리적인 분야로 나눌 수 있는데, 컴퓨터 과학, 경영 과학, 경제학, 심리학, 의사 소통 기법, 문제 해결을 위한 공학적 접근 방법 등의 분야와 관련이 많다.

원래의 소프트웨어 산물에는 물질적인 특성이 없어 그 산물을 눈으로 볼 수 없다. 이러한 소프트웨어 산물을 유형화한 것이 프로그램과 문서인데, 이들은 소프트웨어 산물의 허상에 불과할 뿐 원래의 소프트웨어 산물은 결코 아니다. 프로그램과 문서도 원래의 소프트웨어 산물을 제대로 나타내지 못함은 물론 이들 자체도 그 구성 부분 및 구성부분들 간의 인터페이스(interface)가 대단히 모호하다. 이 때문에 소프트웨어 공학은 가시성과 물질적인 특성을 바탕으로 한 다른 분야의 공학과는 근본적으로 달라, 장구한 역사를 가진 기존의 공학적인 방법을 거의 이용할 수 없다.

현재 거의 모든 소프트웨어 개발 사업이 기간 내 수행, 자원 내 수행, 요구사항의 충족, 사용성, 신뢰성, 유지보수성 등의 제 문제에 부딪혀 많은 애를 먹고 있다. 그 이유는 소프트웨어 공학의 발전이 미미한 때문인데, 기술적인 문제들과 관리적인 문제들이 함께 내재하고 있다. 기술적인 문제들은 어느 정도 연구가 되어 그런대로 성과를 내고 있는 편이나 관리적인 문제들에 대한 연구는 부족하여 소프트웨어 위기(software crisis)의 대처에 약간 부족함이 있는 것이 현재의 상황이다. 더욱이 소프트웨어에 대한 요구가 앞으로 급격히 늘어날 것이므로, 소프트웨어 공학의 현저한 발전이 없으면 소프트웨어 위기에 직면할 수도 있다.

2 소프트웨어 공학의 변천

소프트웨어의 발전 과정은 컴퓨터 시스템의 발전 과정과 밀접한 관계가 있다. 하드웨어의 성능이 좋아지고, 크기가 작아지고, 가격이 저렴해지는 데 따라 다양한 컴퓨터시스템의 출현을 재촉하였다. 소프트웨어도 이에 따라 꾸준히 발전하여 왔기 때문이다. 소프트웨어의 발전 과정을 컴퓨터 시스템의 발전과 연계하여 1950년대부터 10년 단위로 그 개요를 설명하면 다음과 같다. 그 동안의 발전 과정에서 소프트웨어에 관련된 중요한 사항들이 많이 일어났지만, 여기에는 소프트웨어 공학을 이해하는 데 필요한 사항 위주로 설명한다.

1950년대는 소프트웨어의 원시 시대로 소프트웨어 개발에 대한 관리가 사실상 없었다. 하드웨어의 중요성에 가리어져서 소프트웨어는 항상 뒷전이었으며, 소프트웨어는 필요할 때마다 주문하는 형식으로 만들어지고 배포 또한 극히 제한적이었다. 1960년대는 소프트웨어의 생성기로 소프트웨어에 대한 인식이 높아져서 소프트웨어의 제품화가 이루어지고 소프트웨어 하우스(software house)가 등장하여 소프트웨어의 생산품이 널리 보급되기 시작하였다. 이로 인하여 소프트웨어 유지보수 비용이 급격히 증가하기 시작하여 마침내 소프트웨어 위기의 조짐이 보이기 시작하였다.

1970년대는 소프트웨어의 기법 시대로 소프트웨어를 체계적으로 개발하려는 움직임에 따라 여러 가지 소프트웨어 개발 기법이 나타나기 시작하였다. 하드웨어의 비용저하가 두드러지게 나타났으며, 이에 따른 소프트웨어 요구가 급격히 증대되어 소프트웨어의 중요성이 더욱 강조되기 시작하였다. 소프트웨어 위기 현상은 심화되었고 소프트웨어 공학이 태동하였다. 1980년대는 소프트웨어 공학의 중흥기로 소프트웨어공학에 대한 새로운 인식과 소프트웨어의 자동 생산을 위한 기법이 연구되고 인공 지능(artificial intelligence) 분야의 발전이 진행되었다. 또 하나의 특징은 소프트웨어의 유지보수비용이 개발비용을 훨씬 능가하는 것이다. 1990년대는 1980년대의 상황이 심화되는 추세를 보이고 있다.

3 소프트웨어 공학의 특성

소프트웨어 공학으로 나아가려면 소프트웨어의 특성을 파악하여 이 특성에 맞는 방법으로 접근하는 것이 좋다. 우선 소프트웨어의 특성을 분석하는 것이 첫 출발이 되는데 소프트웨어에는 다음과 같은 특성이 있다.

첫째, 소프트웨어는 하드웨어와 같이 물질적인 형태를 취하고 있지 않으므로, 소프트웨어의 특성은 물질적인 것에 바탕을 둔 하드웨어의 특성과 완전히 다르다.

둘째, 소프트웨어 제품은 공학적으로 만들어지는 것이지 하드웨어 제품과 같이 제작되는 것이 아니다.

셋째, 소프트웨어 제품은 노후화하지(wear out) 않으며 다만 쓸모가 없게 되거나 품질이 저하된다.

넷째, 소프트웨어의 유지보수는 하드웨어와 달리 많은 경우 설계의 변경이 요구된다.

종합하여 좀더 자세히 소프트웨어 공학에 중요한 요소들을 설명하면 다음과 같다. 어떤 제품의 사용성이 이 제품의 쓸모와 품질에 좌우되는 것은 소프트웨어의 경우도 예외가 아니다. 쓸모가 유지되려면 제품이 계속적으로 보완되고 향상되어 주위 환경의 변화에 능동적으로 대처할 수 있어야 한다. 이 때 변경 통제(change control) 를 철저히 하여 제품의 형상(configuration)을 제대로 유지하여야 형상 변경에 따른 품질의 저하를 최소로 할 수 있다. 소프트웨어는 하드웨어와 달리 물질적인 형태를 취하고 있지 않으며 소프트웨어의 변경에는 많은 경우 설계 변경이 요구되므로 형상의 유지가 매우 어렵다. 그림 1과 그림2를 비교해 보면 이것을 잘 알 수 있다.

하드웨어의 신뢰도 곡선은 그림 1과 같이 안정

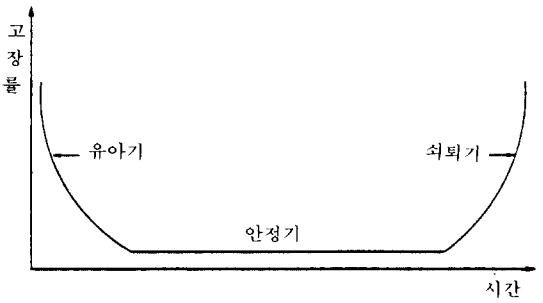

그림 1. 하드웨어 제품의 신뢰도 곡선

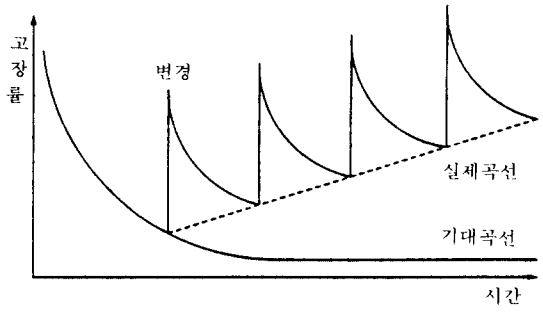

그림 2. 소프트웨어 제품의 신뢰도 곡선

될 때까지 비교적 고장이 많은 유아기를 거치고 다음에 비교적 긴 안정기를 거쳐 쇠퇴기에 이르는데, 쇠퇴기에 이르면 그 하드웨어는 수명을 다하게 된다. 소프트웨어의 경우에는 비슷한 형태의 유아기를 거치는 것은 하드웨어와 같으나, 유지보수 단계가 되어 변경이 시작되면 그림 2와 같은 형태로 품질이 저하되는데 어느 정도 이상 품질이 저하되면 그 산물은 사용이 불가능하게 된다. 저하되는 속도는 형상 관리(configuration control)의 정도에 따라 차이가 있어 결국 소프트웨어의 수명은 형상 관리에 달려 있는 셈이다. 그러므로 소프트웨어 유지보수에는 통상적인 유지보수 활동을 위해서 뿐만아니라 효율적인 형상 관리를 위해서도 개발에 직접 참여한 기술자가 많이 참여하여야 한다.

소프트웨어는 컴퓨터에 존재하며 항상 컴퓨터 하드웨어와 함께 존재한다. 컴퓨터도 단독으로 있는 경우와 큰 시스템에 내장되는 경우가 있는데, 두 경우가 인터페이스 방식은 다른 점이 많으나 소프트웨어 공학의 관점에서는 다를 것이 없다. 그러므로 소프트웨어 공학에는 소프트웨어 단독의 경우와 하드웨어와 함께 존재하는 경우의 두 가지만 고려하면 된다. 먼저 소프트웨어 단독의 경우는 소프트웨어 자체가 시스템을 구성하며 소프트웨어 공학의 외부 환경은 주위 환경이 된다. 다음은 소프트웨어가 하드웨어와 함께 있는 경우로 이 때 소프트웨어 공학은 컴퓨터 시스템공학(computer system engineering)의 한 부문이 되며, 따라서 소프트웨어 공학의 외부 환경은 주위 환경이 아니고 시스템이 된다.

이러한 내용을 그림 3을 통하여 좀더 자세히 설명하면 다음과 같다. 그림 (a)는 소프트웨어 단독의 경우이고 그림 (b)와 그림 (c)가 하드웨어와 함께 존재하는 경우이다. 그림 (b)는 먼저

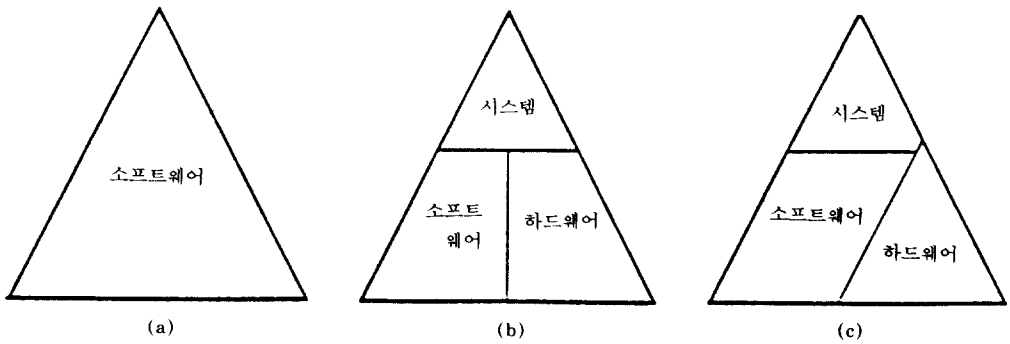

그림 3. 소프트웨어 공학의 존재 방식

시스템 공학이 이루어진 후에 소프트웨어 공학과 하드웨어 공학이 동시에 이루어지는 경우이고, 그림 (c)는 시스템 공학이 먼저 소프트웨어 위주로 이루어진 후에 소프트웨어 공학과 하드웨어 공학이 이루어지는 경우이다. 통합 과정도 비슷한 방식으로 설명된다. (b)의 방식은 상황 전개가 명료해 보여서 이것이 장점으로 보이나, 실제 사업수행에서 항상 하드웨어와 소프트웨어를 이처럼 대등하게 분리해야 하며 이를 수행하는 담당 조직도 항상 대등하게 분리하여야 하는 어려움이 있다. 또 개발시에 하드웨어가 소프트웨어만큼 시스템과 이처럼 밀접하지 않다는 등의 문제점이 있다.

(c)의 방식은 (b)의 방식만큼 명료하지 못하며, 하드웨어가 소프트웨어를 보조하는 인상을 받을 수 있으므로 하드웨어 분야의 사람들이 좋아하지 않는 등의 결점이 있다. 반면 실제 사업 수행은 대부분 (c)의 방식으로 진행되기 때문에 이것이 큰 장점이다. 이 경우 상위 부분에 해당되는 정의와 개발의 일부는 시스템 차원에서 수행하고 대부분의 개발은 소프트웨어와 하드웨어로 구분하는 것이 일반적이다. 여기서 흥미 있는 점의 하나는 이 때 소프트웨어는 하드웨어와 달리 시스템 부문과의 경계에서 넓게 접하고 있으며 그 경계에 융통성이 많다는 것이다.

소프트웨어는 무게, 크기, 색깔, 냄새 등이 없는 무형의 인공물이다. 이러한 소프트웨어 산물을 다루는 소프트웨어 공학은 다른 공학들과는 여러 가지 면에서 근본적인 차이점이 있다. 이 차이점의 근원은 소프트웨어의 본질에서 기인하는 것으로 소프트웨어 산물에는 물질적인 법칙이 적용되지 않고, 산물을 눈으로 볼 수 없으며, 산물의 구성부분들(components)간에 접속이 모호하다는 것 등이다. 우리 주위에 소프트웨어 공학을 이해하는 사람은 별로 많지 않으며, 이것을 실천할 수 있는 사람은 더욱 드물다.

4 소프트웨어 공학의 내용

소프트웨어 공학의 주요 내용을 크게 분류하면 정의, 개발, 유지보수, 사업 관리, 품질 보증, 개발 환경 등이 된다. 여기서 정의, 개발 및 유지보수는 라이프 사이클의 단계들이며, 사업 관리와 개발 환경은 제 단계의 관리와 지원에 관한 사항이다. 정의, 개발 및 유지보수는 소프트웨어의 일생을 나타내고 있는데, 이것을 자세하게 나타내면 소프트웨어 개발 사업에서 채택하는 개발 방법론(development methodology)에 따라 달라진다. 소프트웨어 공학에서는 소프트웨어의 처리 단계 뿐만 아니라 제 단계의 관리는 물론 품질 보증도 매우 중요하며, 또한 이들을 효율적으로 지원할 수 있는 개발 환경도 매우 중요한 의미를 갖는다.

소프트웨어 공학은 단독으로 존재하는 경우와 컴퓨터 시스템 공학의 한 부문이 되는 경우가 있다. 단독으로 존재하는 경우에 소프트웨어 공학의 외부 환경은 주위 환경이 되며 소프트웨어를 시스템과 동일하게 취급한다. 컴퓨터 시스템 공학의 한 부문이 되는 경우에 소프트웨어 공학의 외부 환경은 컴퓨터 시스템이므로 소프트웨어 공학의 내용에 이점을 반영하여야 한다. 이 경우 라이프 사이클(life-cycle)의 단계 중에서 정의는 시스템차원에서 수행하고 개발은 소프트웨어와 하드웨어로 구분하는 것이 일반적인 방식이나, 컴퓨터 시스템 공학을 먼저 수행한 후 소프트웨어와 하드웨어를 구분하는 방식도

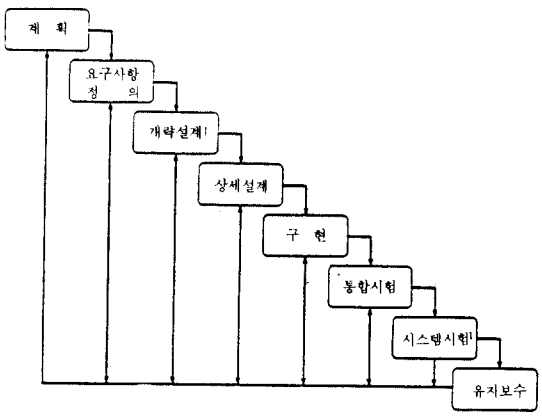

그림 4. 전형적인 라이프 사이클 모델

있다.

라이프 사이클은 사업의 시작부터 끝까지 따라야 할 단계와 그에 따른 사업의 형태 변화를 말한다. 개발 방법론에 따라 여러 가지 형태가 있으나 정의, 개발 및 유지보수의 범위 내에서 주요 활동과 그에 따른 산물로 표현된다. 전형적인 라이프 사이클 모델을 그림으로 나타내면 그림 4와 같다.

정의 단계에서는 앞으로 만들 소프트웨어 산물이 무엇을 할 것인지에 중점을 두고, 개발 단계에서는 그 산물을 어떻게 구현할 것인지에 중점을 두며, 유지보수 단계에서는 그 산물의 변경에 중점을 둔다. 각 단계는 보다 자세하게 세분되어 사업에서 채택하는 개발방법론에 적용되는데, 여기서는 개발 방법론에 독립적인 설명을 위하여 구성 부분들만 나타낸다.

정의의 주요 구성 부분은 사업 계획과 요구사항이다. 사업 계획은 작업범위(work scope), 개발 과정, 조직, 비용, 자원, 시간 계획(schedule), 기법, 도구 등 사업에 관련되는 거의 모든 것을 포함한다. 대부분의 경우 지나친 계획을 하거나 부족한 계획을 하게 되는데 지나친 계획이 부족한 계획보다는 덜 위험하다. 대규모 사업이 실패하는 주된 원인이 바로 부족한 사업 계획 때문이다. 요구사항(requirements)의 목적은 사용자의 운용에 따른 영향의 관점에서 사업 계획에서 제안된 작업 범위에 대한 상세한 정의를 내리는 것이다. 이 요구사항에는 사용자가 보는 모든 것을 포함하여야 하나 사용자에게 보여지지 않는 것을 포함하는 것은 피해야 한다.

개발은 소프트웨어 라이프 사이클에서 통상적으로 가장 중요시하고 가시적으로 자원이 가장 많이 투입되는 부분이다. 요구사항이 정의되면 개발이 시작되는데 요구사항에서 정의된 내용을 설계하고 구현하는 것이 개발이다. 개발은 개략 설계, 상세 설계, 코딩 및 시험을 포함한다. 개략 설계는 소프트웨어의 구조를 만드는 단계인데 소프트웨어 산물을 구성하는 중요한 부분들과 그들 간의 관계를 정의한다. 상세 설계는 개략 설계와 관련하여 구현 단위의 구체적인 절차를 정의하는 것이다. 설계의 구성을 그림으로 나타내면 그림 5와 같다. 코딩은 채택된 프로그래밍 언어(programming language)를 이용하여 이 처리 절차에 상응하는 프로그램을 만드는 것이며, 시험은 단위시험(unit testing), 통합 시험(integration testing), 검증 시험(validation testing) 및 시스템 시험(system testing)을 준비하고 수행하는 것이다.

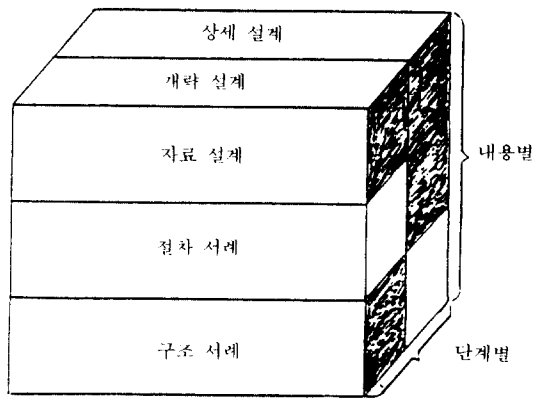

그림 5. 설계의 구성

유지보수는 정의와 개발 단계에서 만들어진 소프트웨어 산물이 사용자에게 인도된 다음에 이루어지는 소프트웨어 산물에 대한 수정과 보완을 뜻한다. 실질적으로 소프트웨어 라이프 사이클에서 가장 많은 비용과 노력이 소요되는 부분이나 현실적으로 매우 많은 문제점과 문제점 해결에 소요되는 비용과 노력이 표면에 노출되지 못하고 있다. 그러므로 흔히 유지보수를 빙산에 비유한다. 유지보수의 성패는 이 단계의 활동에 있는 것이 아니라 최초에 시스템이 정의되고 개발되는 단계로부터 유지보수성에 대한 고려에 있다. 유지보수 단계에서 아무리 유지보수를 잘하여도 이미 만들어진 소프트웨어 산물의 품질 이상으로 품질 향상을 기할 수 없다.

사업 관리의 목표는 전체 라이프 사이클을 통하여 소프트웨어 개발 사업의 가시성(visib-

ility) 을 향상시켜 효율적인 관리를 가능하게 하는 데 있다. 사업 관리는 사업계획과 사업에서 채택한 개발 방법론을 바탕으로 사업 수행의 원칙에 따라 사업을 관리하고 평가하며 통제하는 활동을 말한다. 품질 보증(quality assurance)은 산물의 품질이 일정한 기준에 부합되는지를 확인하기 위해 수행되는 활동을 말한다. 이들은 검토 절차, 표준화, 형상 관리, 방출 절차, 개발 환경 등과도 밀접한 관계가 있다. 검토 절차는 검토에 대한 절차, 표준화는 개발 방법론, 문서, 코딩 등에 대한 표준, 형상 관리는 형상의 변경에 대한 관리, 방출 절차는 방출에 대한 절차를 말한다.

개발 환경의 기본은 각 단계의 자동화를 지원하는 도구(tools)이다. 도구들은 일반적으로 상호간에 관계가 없이 개별적으로 사용되는 경우가 많다. 이러한 도구들이 장비와 융합되어 라이프 사이클의 모든 활동을 지원하기 위하여 통합된 것이 개발환경이다. 흔히 이것을 CASE(Computer-Aided Software Engineering)라고 현재 부르기도 한다. 좋은 개발 환경은 간단하고 일관성이 있으며 사용자에게 친숙한(user friendly) 인터페이스를 제공하므로 배우기 쉽고 사용이 간편하다. 개발 환경의 유형에는 언어 중심 환경, 구조 지향 환경, 도구 키트 환경, 기법 기반 환경 등이 있다.

5 결 어

소프트웨어 공학은 사용성, 신뢰성, 유지보수성 등을 갖춘 소프트웨어를 경제적으로 획득하기 위하여 공학적인 원칙을 확립하고 사용하는 가술이다. 소프트웨어에는 물질적인 특성이 없어 그 산물을 눈으로 볼 수 없다. 그러므로 소프트웨어 공학은 가시성과 물질적인 특성을 바탕으로 한 다른 분야의 공학과는 근본적으로 달라 기존의 공학적인 방법을 거의 이용할 수 없다.

소프트웨어 공학의 앞날을 생각해 보면 소프트웨어 공학의 앞길이 무척 험하게 보인다. 소프트웨어의 유별난 특성에 기인하여 발생하고 있는 문제점들을 이 소프트웨어 공학을 통하여 해결하여야 하기 때문이다. 많은 사람들은 대형 소프트웨어 개발 사업이나 소프트웨어가 관련된 시스템 개발 사업은 거대한 수렁과 같다고 말한다. 한번 빠져 들면 빠져 나오기가 거의 불가능하며 발버둥질을 치며 칠수록 더 빠져 든다는 것이다. 이러한 수렁에 빠져 들지 않을 수 있는 유일한 방책은 소프트웨어 공학을 이해하고 실천하는 것이다. 우리는 이렇게 중요한 소프트웨어 공학이 계속해서 빨리 발전할 수 있도록 모두 노력하여야 할 것이다.

일천한 역사에 비하면 소프트웨어 공학이 그동안 많이 발전한 것은 사실이지만, 앞으로 비약적인 발전을 하려면 지금까지 방법보다 좀 다른 차원의 접근 방법이 필요할 것으로 보인다. 소프트웨어 공학의 지금까지 방법은 현상 유지나 타개책은 될 수 있어도 근본적인 해결책은 되지 못한다. 그 근본적인 해결책의 하나로 소프트웨어의 자동 생산을 들 수 있는데 그 바탕은 아마 인공 지능이 될 것이다. 지금까지 이 분야에 상당한 노력을 기울여 왔으나 실적은 아직 미미한 편이다. 소프트웨어의 자동 생산은 소프트웨어 공장(software factory)의 방식으로 표준화된 공정하에서 소프트웨어를 자동으로 생산할 수 있는 방식일 것이다.

참 고 문 헌

1. F.P.Jr. Brooks, The Mythical Man-Month. Reading : Addison-Wesley, 1979.
2. R.S. Pressman, SOFTWATE ENGINEERING : A PRACTITIONER'S APPROACH. New York : McGraw-Hill, 1987.
3. A.P.Sage and J.D. Palmer, Software Systems Engineering. New York : John Wiley & Sons, Inc., 1990.
4. 천유식, 무에서 무형으로(소프트웨어 공학의 실제). 서울 : 교학사, 1989.
5. 천유식, 시스템 개발 방법론. 서울 : 컴퓨터월드, 1991.

千 柳 植

저자약력

-1969년 서울공대 응용물리학과(학사)
-1982년 동국대학교 대학원 전자계산학과(석사)
-1987년 서울대학교 대학원 계산통계학과(박사, 전산학 전공)
-1961년~ 1976년 KIST, 연구원
-1976년~ 1979년 삼성 GTE통신(주), 실장
-1979년 3월 이후 한국전자통신 연구소,
현재 행정전산망주전산기개발본부 본부장
관심분야 : 컴퓨터 시스템 공학, 소프트웨어 공학, 개발 방법론 등