기 술

'90년대 소프트웨어 개발모델

최 락 만*

〈요 약〉

본 고는 기존 소프트웨어 산업에서의 기술적인 문제점을 검토하고, 생산성 향상을 통하여 소프트웨어 수급격차를 해소하고, 현재의 소프트웨어 위기를 극복할 수 있는 방안으로서 전통적인 개발모델을 기반으로 하는 진화론적인 접근방식과 automation－based 개발모델을 기반으로 하는 혁신적인 접근방식에 대하여 기술한다.

목 차

* 공정제어연구실 실장

I. 서 론

최근 산업사회로 부터 반도체, 컴퓨터 및 통신기술의 발전에 의한 정보화 사회로의 변천에 따라 소프트웨어 제품에 대한 수요는 크게 증가하고 있으나, 공급은 이에 따르지 못하고 있어 소프트웨어의 수급격차는 확대되고 있다. 미국의 경우를 예를 들면, 소프트웨어에 대한 수요는 년 12%씩 증가하고 있으나, 이 분야의 기술 인력과 생산성향상은 각각 년 4%의 증가율에 머무르고 있다. 이에 따라 매년 약 4%의 공급부족이 누적되고 있다.[1]만일, 소프트웨어 기술인력의

증가율이 일정하다고 가정할때, 수요와 공급이 균형을 이루기 위한 생산성 증가율은 1983년을 기준으로 볼때, 1988년까지 2배가 되어야 하며, 1993년까지는 4배가 되어야 할 것이다.

본 고에서는 기존의 소프트웨어 생산 및 이용에서의 기술적인 문제점들을 분석하고, 이와 같은 문제점들을 해결하고, 생산성향상을 도모하여 소프트웨어 수급격차를 해소하고, 현재의 소프트웨어 위기를 극복할 수 있는 방안으로 제시되고 있는 1990년대의 소프트웨어 기술개발 모델을 중심으로 소프트웨어 제조기술의 고도화를 위한 접근방식을 검토하고자 한다.

II. 기존 소프트웨어 산업의 문제점

소프트웨어 산업은 개발 및 생산기술의 발전이 미흡하여 타 산업분야에 비하여 생산성 증가율이 크게 뒤지고 있으며, 기술인력의 부족에 따라 수급격차가 더욱 심화되는 등 소프트웨어 위기를 맞고 있다. 이 절에서는 이러한 수급차질 현상에 대한 원인으로서 주로 기술적인 측면에서 기존 소프트웨어 산업의 문제점으로 개발방법론, 유지·보수실태, 재활용성 및 표준화등에 대하여 분석한다.

1. 개발 방법론상의 문제점

소프트웨어는 일반적으로 사용자 요구분석, 시스템 분석 및 구현방식의 설계, 세부기능설계, 코딩(coding), 모듈 및 통합테스트, 유지·보수와 같은 공정과정을 거치게 되는데, 이러한 개발형식에 기반을 둔 현재의 소프트웨어 개발방식에서 사용자는 인수시험 단계에 이르기까지 개발되고있는 시스템이 자기의 요구를 만족시켜 주는지 알수 없는 경우가 많으며, 인수시험단계에서 발견된 소프트웨어 오류를 수정하기 위해서는 사용자 요구분석 공정부터 재 작업을 해야되는 경우도 있다.

이와 같은 오류발견 시점의 지연은 오류의 올바른 수정을 어렵게 할뿐아니라, 오류수정 소요시간 및 경비를 증가시켜, 결과적으로 시스템에서의 소프트웨어 가격비중을 상승시키는 결과를 초래케 한다. 이러한 문제의 해결을 위해서는 각 공정단계에서 사용자의 요구를 만족하는지의 여부를 검사하여 소프트웨어 오류를 조기에 발견하고 해소시키는 것이 중요하다.

2. 유지·보수상의 문제점

소프트웨어 제품의 사용중 발견된 오류의 수정, 기존 기능의 개선 및 새로운 기능의 추가를 위한 소프트웨어 유지·보수경비는 전체 소프트웨어 가격의 약 70~80%를 차지하고 있다. 이러한 현상의 근본적인 원인은 제품의 신뢰성결여에 기인하지만, 소프트웨어 유지·보수가 이해하기 어려운 원시코드(source code)를 대상으로 하기 때문이다. 즉, 기존 소프트웨어 제품의 유지·보수는 대부분 그 제품의 개발자가 하는 것이 아니고, 이용자가 하는 것으로, 이용자가 원시코드를 읽고 이해하기란 매우 어려울 뿐아니라, 부작용(side-effect)이 발생할 가능성이 높다. 이러한 문제점을 해결하기 위해서는 유지·보수의 대상을 비교적 이해하기 쉬운 각종 규격(사용자 요구규격, 시스템규격, 설계규격등)으로 하여야 하며, 각종 규격은 문서화(documentation)가 잘되어 있어야 한다.

3. 기계처리성(Machine Processability)의 결여

광의적인 의미의 소프트웨어에는 프로그램을 포함하여, 프로그램을 제작하기 위한 각종 규격

및 문서를 포함시킬 수 있다. 이와 같은 소프트웨어 제품중 컴퓨터언어로 작성된 프로그램만 기계처리가 가능할 뿐, 코딩이전 단계의 결과인 각종 규격등은 문서화조차 되어있지 않거나, 문서화가 되어 있어도 자연언어를 사용함으로서그 의미가 애매모호하고, 불완전하며 모순투성인 조잡한 규격이 작성되는 경우가 대부분이다. 이러한 규격을 기반으로 하여 제작되는 프로그램은 사용자의 요구를 만족시키지 못하는 결과를 낳게 된다. 따라서, 코딩이전의 각종 규격을 형식성(formality)이 있는 언어를 사용하여 기술하고, 기계처리에 의해 완전성(completeness), 무모순성(consistency)등 제반 품질특성을 검사한다면 소프트웨어의 품질을 제고시킬 수 있다.

4. 재 활용성 문제

소프트웨어 제품, 특히 대형·복잡한 소프트웨어 제품의 경우, 기본기능은 대부분 동일하지만, host machine의 기종이 다르거나, 응용대상에 약간의 차이만 있어도 기존 제품의 재활용이 불가능하여 재개발하여야 할 경우가 많다. 이러한 문제는 이해하기 어려운 원시코드를 재활용 대상으로 한다는데 기인하는데, 그 대상을 컴퓨터언어보다는 자연언어에 더 가까운 고급언어를 사용하여 기계처리가 가능한, 우수한 품질의 규격을 대상으로 한다면 소프트웨어 제품의 재활용성은 향상될 것이다. 이때, 규격을 원시코드로의 자동변환이 가능하게 하면, 재활용성을 크게 증가시킬 수 있을 것이다.

5. 표준화의 결여

소프트웨어 생산은 아직도 노동 집약적인 형태를 크게 벗어나지 못하고 있으며, 실용적인 개발모델조차도 확립되어 있지 못한 실정이다. 또한, 소프트웨어는 그 특성상 용어의 표준화나 방법론의 규범화 이외의 표준화가 대단히 어렵지만, 소프트웨어를 개발할때 동일한 개발도구나 규범화된 방법론을 사용하면, 소프트웨어 제품의 표준화는 어느 정도 달성이 가능하다. 그러나, 현재의 시점에서 실용적인 소프트웨어 개발도구나 방법론은 많지 못하므로, 성능이 우수한 여러가지 용도의 개발도구와 방법론을 개발·보급하는 것이 필요하다.

III. 소프트웨어 기술발전을 위한 접근방식

앞에서 언급한 소프트웨어 산업의 문제점을 극복하고, 생산성 향상을 통한 수급격차의 해소 방안으로 현재 제시되고 있는 접근방식으로는 진화론적 접근방식(evolutionary approach)과 혁신론적 접근방식(revolutionary approach)가 있다.[1,2]

1. 진화론적 접근방식

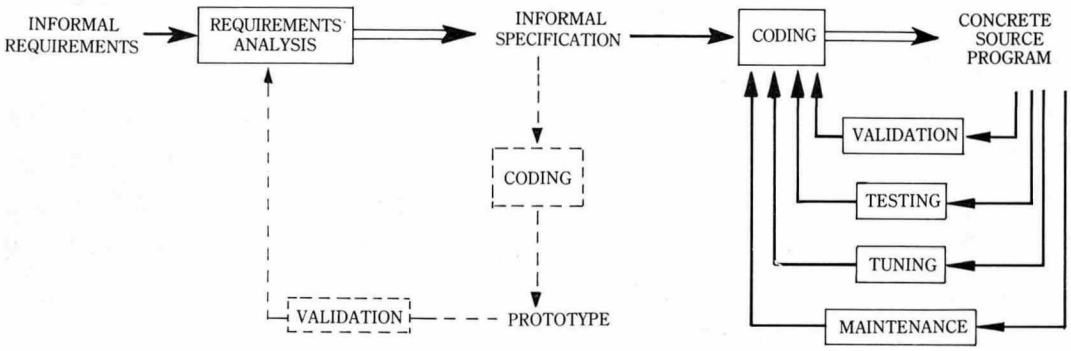
이 방식은 우리가 현재 직면하고 있으며 앞으로 예측되는 문제점들을 극복하고, 수급격차의 해소를 위한 생산성향상 방안으로서의 만병통치약은 과거의 소프트웨어 기술 발전과정이나, 현재의 기술수준으로는 기대하기 어렵다는 판단에 기반을 두고, 기존에 정립된 기술들을 기초로하여 소프트웨어 개발기술을 점진적으로 개량·발전시켜 가는 방식이다.

즉, 진화론적인 이 방식은 (그림1)과 같은 전통적인 소프트웨어 개발모델에 기반을 두고서, ① 기존의 개발도구, 방법론, 소프트웨어 개발환경, 교육 및 관리기법등을 개선하고, ② rapid prototyping, 요구공정용도구, 재활용이 가능한 소프트웨어 부품들의 library화등의 새로운 기술개발을 강화시키는 한편, ③ 소프트웨어생산에

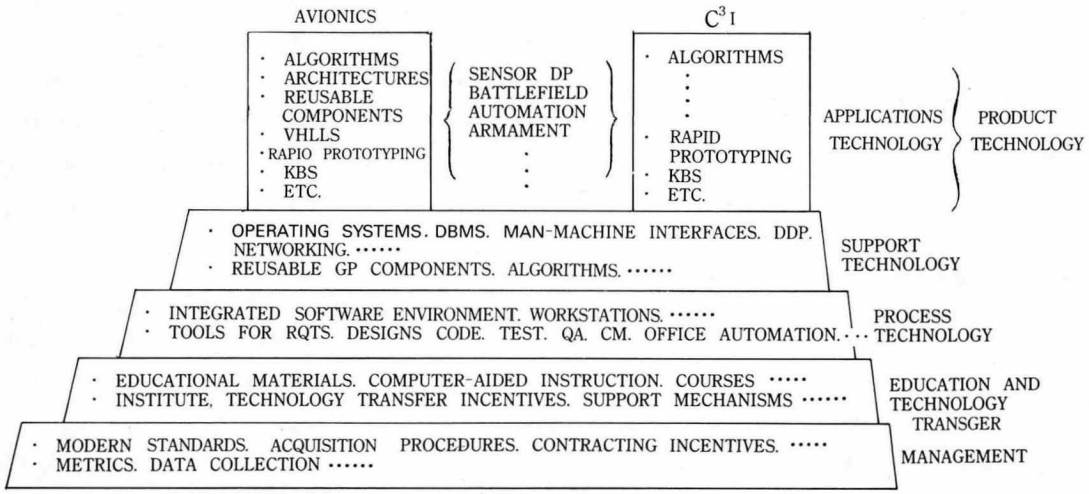
필요한 각종 관리기법, 교육과 기술이전, 공정기술, 지원기술등을 기반으로 하여 각종 생산 응용기술들을 체계적이고, 유기적으로 통합하여 효율적인 소프트웨어 개발환경을 구축하므로서 생산성을 제고하고, 수급격차를 해소시킬 수 있다는 방식으로, (그림2)와 같은 1990년대의 통합화된 소프트웨어생산 기반구조를 제시하고 있으며, 소프트웨어 생산 공통기반구조 (shared infra-structure)와 부가가치 항목(value-addedissues)의 관점에서 볼때 2000년대까지 컴퓨터산업의 발전추이를 〈표1〉과 같이 추정하고 있다. 이와 같은 방식의 대표적인 예로는 미국방성의 STARS(Software Technology for Adaptable, Reliable Systems) 계획이 있다.

2. 혁신론적 접근방식

다른 하나의 접근방식은, 전통적인 개발모델에 기반을 둔 진화론적 접근방식이 생산성을 향상시켜 어느정도 수급격차를 줄일 수 있으나, 여전히 노동집약적이며, 소프트웨어 유지·보수가 원시코드를 주 대상으로 한다는 점, 그리고



(그림 1) 전통적인 스프트웨어 개발모델



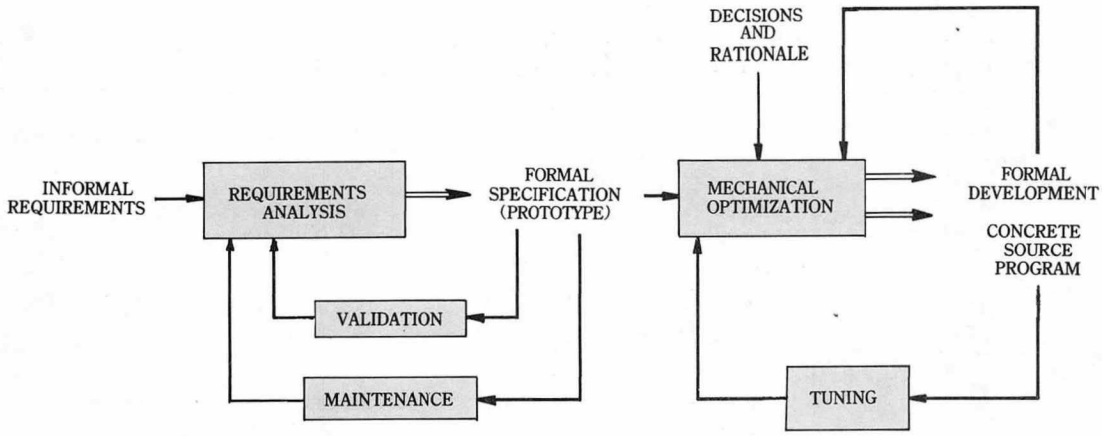
(그림 2) 1990년대의 통합 소프트웨어 기반구조

〈표 1〉 컴퓨터산업의 발전단계

STAGE/DECADE	SHARED INFRASTRUCTURE	VALUE-ADDED ISSUES
I 1950	None	Shared Utilities Media Standards
II 1960	Shared Utilities Media Standards Algorithms	MOL/HOL Mixed Peripherals I/O Standards
III 1970	HOLs Vendor Utilities I/O Standards Plus Previous Base	Software Unbunding Stable OS Interface
IV 1980	Vendors-Line OS. Utilities PlugCompatible Mainframes. Commercial Software Packages. Basic Software Environments. Plus Previous Base	Portable OS Environment Networking Standards
V 1990	Ada Portable OS. Utilities Portable Environments Net-working Standards Some Main-frame Standards Nebula. 1750	Application Standards Mainframe Standards
VI 2000	Knowledge ; Based Application Standards. Program Generators. Component Libraries for Some Areas	Application Standards for More Complex Knowledge Domains

코딩이전 단계의 소프트웨어(예를 들면 기술규격)들이 형식성이 없다는 문제점들을 크게 개선하지 못하고 있어, 수급격차 문제를 해소하기에는 근본적인 해결책이 되지 못하고 미흡한 것으로 판단하고, 소프트웨어 개발기술을 보다 혁신적으로 개선할 필요가 있다고 주장하고 있다.

즉, 혁신론적인 접근방식에서는 요구공정 단계(requirement phase)에서 부터 기능요구와 성능요구가 기술되는 요구명세(requirement specification)를 컴퓨터언어가 아닌 일반 사용자도 이해하기 쉽도록 자연언어에 가까우면서도 기계처리가 가능한 형식성을 갖는 기술언어로 표현하고, 이 형식화된 규격(formal specification)을 기계처리에 의해 validation작업을 반복적으로 수행하므로서 사용자의 요구를 만족시키는 시제품(prototype)을 조기에 만들수 있다는 것이다. 이러한 형식화된 규격은 기계처리가 가능하기 때문에 원시코드로의 자동변환이 가능케 할 수 있으며, 유지·보수공정 또한 그 대상을 원시코드가 아닌 보다 이해하기 쉬운 규격을 대상으로 하므로서 소프트웨어의 유지·보수를 보다 용이하게 할 수 있다는 개념으로, (그림3)과 같은 automation-based 개발모델을 제시하고 있다. Automation-based 개발모델을 (그림1)의 전통적인 개발모델과 비교하면 그 차이점은 〈표2〉와 같다. 이와 같은 automation-based 개발모델에 기반을 둔 접근방식의 예로서는 AT&T의 PAISLey(Process-oriented, Application, and Interpretable Specification Language) Project,[4,6] HOS사의 USE.IT,[3] Teledyne Brown Engineering사



(그림 3) Automation based 소프트웨어 개발모델

〈표 2〉 전통적인 개발모델과 Automation-based 개발모델의 비교표

전통적인 개발모델	Automation-based 개발모델
Informal specification	Formal specification
Prototyping uncommon	Prototyping standard
Prototype created manually	Specification is the prototype
Code validated against intent	Prototype validated against intent
Prototype discarded	Prototype becomes implementation
Manual implementation	Machine-aided implementation
Code tested	Testing eliminated
Source code maintained	Formal specification maintained
Design decisions lost	Automatically documented
Maintenance by patching	Maintenance by replay

의 TAGS (Technology for the Automated Generation of Systems),[5]GTE Lab.의 RPS(Requirements Processing System)[9]등을 들 수 있다.

IV. 결 론

진화론적 접근방식은 전통적인 개발모델을 기반으로하고 기존의 개발기술을 통합하고 점진적으로 개량·발전시켜 가는 방식으로, 보다 현실적이지만 이러한 방식만으로 현재의 소프트웨어 위기를 극복하고 수급격차를 해소할 수 있을 것인지에 대해서는 회의적인 의견이 많다.

이에 비해, 혁신적 접근방식의 automation-based 개발모델은 전통적인 개발모델에 비하여 rapid prototyping에 의한 사용자요구의 조기반영, 유지·보수의 용이성, 소프트웨어의 재활용성, specification의 형식화에 의한 기계처리가능성, 문서화 및 표준화 문제를 크게 개선시킬 수 있는 이상적인 방법이라 할 수 있다. 이와 같은 자동화에 기반을 둔 개발도구로서 일부는 실험단계에 있는 것도 있고, 일부는 상품화되어 있는 것도 있지만, 응용범위의 한계성과 formal specification을 원시코드로의 자동변환 문제를 해결하여야 할 것이다.

끝으로, 1996년까지 정보기술분야에서 세계수준의 연구소를 추구하고 있는 우리 연구소[13]와, 2001년까지 정보산업의 선진화를 위한 정부의 과학기술 개발장기계획[10]의 성공적인 추진을 위해서는 automation-based개발모델에 기반을 둔 최신 기법과 도구들을 조기에 도입하고, 이의 심층연구를 통하여 아직도 낙후된 국내 소프트웨어 제조기술의 고도화를 추진해 나가야 할 것이다.

참 고 문 헌

[1] B. W. Boehm, et al., "Software Technology in the 1990's : Using an Evolutionary Paradigm," Computer, Vol. 16, No.11, Nov. 1983, pp30~37.

[2] R. Balzer, et al., "Software Technology in the 1990's : Using a New Paradigm, " Computer, Vol. 16, No.11, Nov. 1983, pp 39~45.

[3] P. Mimno, "A New Technology for mathematically Provable Software, " Computer World, Oct. 11, 1982.

[4] P. Zave, "The Operational versus the Conventional Approach to Software Development, "Commu. ACM, Vol.27, No.2, Feb. 1984, pp104~118.

[5] G. E. Sievert, et al., "Specification-based Software Engineering with TAGS," Computer, Vol.18, No.4, Apr. 1985, pp56~65.

[6] P. Zave, "An Overview of the PAISLey Project" ACM SIGSOFT Software Engineering Notes, Vol.9, No.4, July. 1984, pp 12~19.

[7] H .A .Sutherland, et al., "Control Engineers Workbench-A Methodology for Microcomputer Implementation of Controls," IEEE Control Systems Magazine, Fab. 1985, pp22~26.

[8] 野木兼六, "요구정의 기술의 최근 동향" 정보처리, Vol.27, No.1, Jan. 1986, pp21~30.

[9] A .M .Davis, "Rapid Prototyping Using Executable Requirements Specifications, "ACM SIGSOFT Software Engineering Notes, Vol.7, No.5, Dec. 1982, pp39~44.

[10] 과기처, "2000년대를 향한 과학기술개발 장기계획," 1986. 9, pp87~104.

[11] 과기처, "소프트웨어 기술개발에 관한 연구 (컴퓨터 기술개발중)", 최종연구보고서, 1986. 7, pp937~945.

[12] 최락만, "소프트웨어 요구정의 공정기술에 관한 조사. 분석", ETRI, 연구보고서, 1986 12.

[13] ETRI, "정보기술동향과 우리의 대응책," 1987. 8, pp299~332.