■ 特 輯 · 技 術 解 說

소프트웨어 엔지니어링과 과학적인 접근 방향

李 京 煥
(正 會 員)
中央大學校 電算學科 教授

Ⅰ. 소프트웨어 공학(software engineering)과 소프트웨어 과학(software science)

소프트웨어의 규모와 적용 범위가 방대해 짐에 따라서 소프트웨어 공학에 관한 관심이 날로 고조되고 있다. 소프트웨어 공학이란 소프트웨어를 개발하고 유지 보수하는데 관련되는 모든 문제와 해결 기법을 연구하는 학문이며, 소프트웨어 공학의 궁극적인 목표는 소프트웨어의 신뢰성을 증가시키고, 보다 용이하게 수정될 수 있도록 하는 동시에 소프트웨어 경비(cost)와 복잡성(complexity)을 감소시키는데 있다.[6] 따라서 소프트웨어 공학자들은 경비가 적게 들고 보다 신뢰할 수 있는 소프트웨어를 개발할 수 있도록 프로그램에 관한 제한사항과 프로그램을 작성하는데 필요한 이론적인 기초에 관하여 연구하고 있다.

프로그래밍이 소프트웨어 공학에서 상당히 중요한 요소이기는 하지만 소프트웨어 공학이 프로그래밍이라는 것은 아니다. 물론 컴파일러의 작성자 혹은 오퍼레이팅 시스템의 구현자들이 소프트웨어 공학에서 개발한 기법을 사용하고 있으나 소프트웨어 공학이 컴파일러나 오퍼레이팅 시스템에 관하여 연구하는 학문은 아니다.[15]

소프트웨어 공학은 여러가지의 학문이 관여되는 복합 학문이다. 이것은 알고리즘을 분석하고 증명하기 위하여 수학을 사용하고, 소프트웨어 경비를 견적하고 사용되는 기술의 장단점을 정의하기 위하여 공학을 적용하며, 소프트웨어 요구를 분석하고 소프트웨어 개발에 참여하는 인원과 개발과정을 관리, 통제, 조정하기 위하여 경영학이 활용된다.

소프트웨어 개발은 통상 다음과 같은 두가지로 분류되어 추진된다.[12] 그중의 하나는 한 사람의 프로그래머가 혼자서 소프트웨어 시스템을 설계하고, 구현하고, 테스팅하는 방법으로 비교적 소규모 시스템의 개발에 적용하는 것이고, 또 다른 하나는 여러 사람이 단체로 대형 소프트웨어 시스템을 개발하는 방법이다. 소프트웨어 공학은 두가지중에서 대형 시스템에 보다 더 관심을 집중시키고 있다.

1959년, 뉴욕시에 있는 "Verrazano Narrows Bridge"를 가설하기 위하여 작업을 시작하였을 당시에는 가설경비를 3억2천5백만 달러로 추정하고 완공예정일자를 1965년으로 예정하였다. 그리고 이 교량은 그 당시까지 한번도 시공해 보지 못했던 거대한 부교(suspension bridge)였는데도 불구하고 계획된 예산 범위내에서 1964년 11월에 완공되었다.[7,8] 그렇지만, 지금까지 개발된 소프트웨어 시스템보다 훨씬 거대한 소프트웨어 시스템을 개발할 때는 교량을 가설할 때와 같은 양상은 한번도 경험해 보지 못했다.

소프트웨어는 종종 계획된 기간과 경비보다 지연되고 초과되어 개발된다. 또한 이것은 신뢰성이 없고 상당한 보수경비가 요구된다. 연인원 5000명이 소요된 IBM 360/OS 프로젝트도 몇년간 지연되어 개발이 완료되었다.[3,4] 교량을 가설하는 토목공학에서는 작업량을 정확하게 판단하고 계획된 기간과 예산 범위내에서 완공될 수 있는데 반하여 소프트웨어 공학에서는 왜 불가능한 것일까? 이에 대한 대답은 다음과 같이 요약할 수 있다. 토목공학자가 대형교량을 가설하는데 소요되는 작업량을 판단하는 것은 소프트웨어 공학자가 대형 소프트웨어 시스템에 소모되는 작업량을 판단하는 것보다 훨씬 용이하다는 사실과 오늘날의 소프트웨어 개발 문제는 소형 소프트웨어 시스템을 개발한 경험을 대형 소프트웨어 시스템에 그대로 적용하는데서 파생되고 있다는 사실이다.[15]

이러한 문제점을 해결하기 위하여 Halstead[10,11]는 소프트웨어 과학(software science)이라는 새로운 학문의 필요성을 주장하였다. Halstead의 정의를 빌리면

"소프트웨어 과학은 컴퓨터 프로그램이나 산문처럼 펜으로 작성된 물질에 관한 측정 가능한 특성을 연구하는 학문"이다. 소프트웨어 과학의 목적은 프로그램에게 양적인 측정방법(quantitative measure)을 제시하는데 있다. 이러한 측정은 하나의 프로그램내에 존재하는 연산자(operator)와 피연산자(operand)를 이용하여 프로그램의 길이, 프로그램의 용량(volume), 그리고 프로그램의 수준(level)등과 같은 특성들을 예측할 수 있는 함수(function)를 고안하였다. 또한 그는 이러한 결과들을 인식 심리(cognitive psychology)[5]에 대한 이론과 주장(assertion)으로 사용하였으며, 서로 상이한 프로그램들을 작성하는데 소요되는 시간과 정신노동의 양을 예측할 수 있는 공식을 유도하였다. 이러한 이론에 관하여 독자들에게 도움을 주기 위하여 소프트웨어 과학의 주요 부분에 대하여 간단히 서술하기로 한다. 어떤 프로그램으로 부터 다음과 같은 자료를 축출할 수 있다.

M_1 : 서로 상이한 연산자의 갯수

M_2 : 서로 상이한 피연산자의 갯수

N_1 : 사용되는 연산자의 총 갯수

N_2 : 사용되는 피연산자의 총 갯수

상기와 같은 가장 기초적인 자료로부터 프로그램이 가지는 어휘(vocabulary)의 갯수 M은 $M=M_1+M_2$로 정의되며, 프로그램의 길이 N은 $N=N_1+N_2$로 정의된다. 프로그램의 용량 V는 비트(bit)단위로 측정된다. 이 측정은 프로그램의 길이에 프로그램의 어휘를 N개의 요소로서 각각 상이하게 표현하는데 필요한 비트의 갯수를 곱함으로써 산출되며 공식으로는 $V=N\log_2 M$이 된다. 프로그램의 수준 L은 알고리즘을 얼마나 간결하게 구현하였느냐를 점검하는 측정이다. 프로그램의 수준이 높으면 높을수록 알고리즘의 구현은 프로시듀어 호출(procedure call)로 구성되어 있다는 것을 의미한다. L의 영역은 $0<L\leq 1$이며 공식으로 나타내면 $L=V^*/V$이 되고, 여기에서 V^*는 프로시듀어 호출로 구현된 알고리즘의 용량이다. 주어진 프로그램을 작성하는데 소요된 노력과 시간은 각각 $E=V/L$, $\hat{T}=E/S$이다. 여기에서 E는 주어진 프로그램을 작성하는데 소요된 기초적인 지적 노력(elementary mental discrimination)의 총 갯수로 해석될 수 있다. 그리고 S는 이와 같은 지적노력이 1초당 수행할 수 있는 갯수를 측정한 값이다. 실제로 소요된 시간은 T로 표현할 수 있으나 여기에서는 추정된 값이기 때문에 $\hat{T}$로 표시한다.

Halstead가 이와 같은 공식을 제한한 이래로 Baker,[1] Gorden[9], Woodfield[14] 등에 의하여 소프트웨어 과학에 대하여 연구가 활발하게 진행되고 있으며, 이들은 대부분이 Halstead가 제안한 공식의 타당성을 점검하기 위하여 실험적인 연구를 계속하고 있다. 또한 Ottenstlin[13]는 개발된 소프트웨어가 테스팅(testing)과 통합(integration)되기 전에 존재하는 결함(bug)의 갯수를 예측하기 위하여 소프트웨어 과학적인 접근방법을 택하였다. 특히 신뢰성의 측정분야에서 이러한 이론이 성숙되어 가고 있다. 시스템 엔지니어링, 프로젝트 관리, 소프트웨어의 재구성 및 소프트웨어 엔지니어링의 기술을 평가하는데 신뢰성 이론이 발전되어 가고 있다. 신뢰성 연구를 위해서는 수학, 경영학등의 이론적인 배경을 쉽게 이용할 수 있기 때문이다. 그리고 프로세스 엔지니어링분야에서는 추상적 데이타 타입에 관한 이론을 정립시키려는 노력들이 많아지고 있다.

Ⅱ. 소프트웨어의 프로세스 엔지니어링

공학의 발전은 과학적인 이론에 근거를 두고 있지만 소프트웨어 엔지니어링은 소프트웨어를 개발하기 위해서 여러가지 공학적인 기법을 동원하여 생산성을 향상시키고 품질을 보증할 수 있는 방법론을 연구하고 툴을 제작하여 자동화된 개발공정을 만들어 낸다. 결국은 경제적인 효율을 고려하여 신뢰성 있는 제품을 생산하기 위해서 지원하는 모든 공학적인 방법론과 툴의 개발 및 사용이 소프트웨어 엔지니어링의 중요한 과제이다.

소프트웨어 양식으로부터 실행을 위한 소프트웨어 개발을 하기위해 活動(activity)의 집합을 소프트웨어의 process engineering이라고 말한다.

프로세스 엔지니어링은 소프트웨어 양식에서부터 설계, 코딩 및 테스트의 단계별 중간제품들을 실행할 컴퓨터와 연관지어 개발할 活動들을 정의하고 각 活動에 적용할 기술과 툴을 선택해 나간다. 각 단계별 活動의 프로세스 엔지니어링의 적용은 같은 방법으로 분석하고 최적화 시켜 나간다.

분석대상은 시간과 공간에 관련된 다음과 같은 속성들이다.

(1) Event의 순서와 타이밍 : 조건별 처리, 동시성 콘트롤, predecessor/successor조건, interrupt handling 등이 실행할 컴퓨터에 맞도록 최적화 되어야 한다.

(2) 물리적 요소의 최적하게 사용 : 화일의 분할사용, 커뮤니케이션, 최적한 자료구조 및 알고리즘을 공유할

수 있는 프로그램을 찾아서 경제성을 높인다.

(3) 결합의 방지와 고립된 프로세스의 처리와 화일의 분해를 통한 안전관리 방법

(4) 프로세스의 구성을 통해서 기능적인 요구를 만족시킨다.

(5) 수정보완을 효율적으로 수행할 수 있게 한다.

이와 같은 항목을 중심으로 하여 분석하여 적당한 방법론을 도입하여 소프트웨어를 개발하고 개발된 소프트웨어가 요구분석에서 정의된 기능을 가지고 있는가? 그리고 프로그램의 내부구조가 복잡도나 structureness 상으로 만족스러운가를 평가하는 活動이 프로세스 엔지니어링의 중요한 과제이다. 유지보수를 효율화시키는 방법과 함께 소프트웨어의 부품화를 추구하기 위한 소프트웨어의 재구성 방법도 개발 초기부터 고려함으로써 다음에 생산될 제품의 경제성을 평가할 수 있도록 엔지니어링 기술이 발전되고 있다.

이렇게하여 개발된 소프트웨어는 사용자 매뉴얼과 유지보수 매뉴얼을 작성함으로써 최종제품이 될 수 있다. 이러한 일련의 活動을 지원하기 위해서는 표준화된 방법론과 툴을 어떻게 도입할 것인가를 연구하고 실제로 적용하는 것이 곧 소프트웨어 엔지니어링이다.

이러한 프로세스 엔지니어링을 소프트웨어의 개발활동에 어떻게 적용할 것인가를 생각해 보자.

Ⅲ. 소프트웨어 공학의 활동(activity)

앞에서도 언급한 바와 같이 소프트웨어 공학을 적용하는 방법은 크게 두가지의 카테고리(category)로 분류될 수 있다.[12] 그중의 하나는 개별 프로그래머들이 적당한 크기의 프로그램(통상, 프로그램 문장의 갯수가 수천이내)을 짧은기간(통상, 몇개월이내)에 완성시키는 소형 소프트웨어의 개발작업(programming-in-the-small)과 여러 사람들이 개발팀을 형성하여 거대한 작업을 장기간 추진하는 대형 소프트웨어 개발작업(programming-in-the-large)으로 구분할 수 있다. 따라서 적용방법에 따라서 소프트웨어 공학 활동은 다음과 같이 상이할 수 있다.

1. 소형 소프트웨어 개발 작업에서의 활동

소형 소프트웨어를 개발하는 작업에 대한 하나의 예는 원유 채유 장비(oil well logging tool)를 들 수 있다. 이 장비는 원유에 포함되어 있는 여러 가지의 물리적인 특성과 원유를 둘러싸고 있는 암석은 어떻게 형성되어 있으며 채유될 수 있는 원유가 얼마나 남아 있는지를 추정하기 위하여 사용된다. 채유장비는 장거리 케이블을 이용하여 원격지에 있는 컴퓨터와 연결되며, 컴퓨터에 의하여 제어된다. 채유장비의 동작을 제어하기 위하여 소프트웨어는 채유장비의 감지장치(sensor)로부터 수집된 자료를 차후에 분석하기 위하여 계속 저장해 두어야 한다.

채유장비를 위한 소프트웨어는 일반적으로 수천개의 프로그램 문장으로 구성되어 있으며, 개발하는데 통상 수개월 내지 1년 정도의 기간이 소요된다.

우리는 새로운 채유장비가 개발되면 이 장비에 적합한 소프트웨어를 개발하지 않으면 안된다. 따라서, 새로운 소프트웨어를 개발하기 위해서는 일반적으로 다음과 같은 절차를 밟게 된다.

(1) 소프트웨어 사양서(specification) 작성

소프트웨어 사양서는 소프트웨어가 무엇을 해야 하는지를 서술한 문서를 말한다. 따라서, 먼저 채유장비에 대한 연구와 이 장비가 무슨 측정이론을 근거로 하고 있는지를 연구해야만 한다. 그러므로 이 장비를 설계한 설계 담당자와 광범위한 대화가 이루어져야 한다.

(2) 사양서 분리(decomposition)

위에서 작성한 사양서를 좀 더 취급하기 쉽도록 하기 위하여 상대적으로 독립된 여러가지 조각으로 분리한다. 이 행동은 사양서에 대한 지식이 절대적으로 요구된다. 또한 프로그램 구성에 관한 기술과 소프트웨어가 사용될 컴퓨터의 특성에 관한 지식도 필요하다. 사양서 분리 작업은 사양서 내용의 애매모호한 사항과 상호 모순되는 점을 제거하고 사양서에 누락된 정보를 보강하기 위하여 채유장비 설계 담당자와 계속 반복하여 상의하여야 한다.

(3) 구현(implementation)

위에서 분리한 각 요소에 대하여 코딩작업을 실시하는 단계이다. 따라서 프로그래밍 기술과 사용할 컴퓨터에 관한 지식이 필요하다. 또한 이 단계에서도 장비설계 담당자와 계속 반복하여 의견을 교환해야 한다.

(4) 테스팅(testing)

이 단계에서는 소프트웨어 사양서에 입각하여 정확하게 구분되었는지를 테스트 한다. 따라서 사양서와 채유장비에 관하여 상세히 알고 있어야 한다.

(5) 최적화(optimization)

컴퓨터는 감지장치로 부터 통신시스템을 통하여 입수되는 자료를 연속적으로 기록해야 하기 때문에 실시간 시스팀(real-time system)이 되어야 한다. 따라서, 이러한 제한사항을 가장 적절하게 만족하는지를 점검하고 미비한 점은 보강되어져야 한다. 이것은 대량의 측정과 테스팅이 수반되기 때문에 상당한 시간이 소

요되는 작업이다. 또한 채유장비의 상세한 구성 및 특성, 그리고 측정이론에 대하여 알고 있어야 하기 때문에 장비 설계자의 조언이 필요하다.

(6) 검증(validation)

이 단계에서는 개발된 소프트웨어가 실제로 원하는 소프트웨어라는 것을 확인한다. 개발자의 막대한 노력에도 불구하고 소프트웨어를 채유장비에 결합시켰을때 원하는 결과가 산출되지 않는 경우가 종종 있게 된다. 이것은 채유장비가 어떻게 작동되고 있는가에 대하여 잘 인식되지 못함으로 부터 기인된다. 따라서 사양서와 프로그램은 반드시 수정되어야 하고 테스팅이 동반되어야 한다.

(7) 개선(evolution)

채유장비가 개선되고 새로운 측정이론을 적용함에 따라서 소프트웨어는 계속 개선되어야 한다. 이와 같은 활동을 위해서는 채유장비와 측정이론, 그리고 소프트웨어가 어떻게 구현 되었는가에 대하여 알고 있어야 한다. 따라서 이 작업은 소프트웨어 개발에 관여한 여러 사람들과 상의하여야 한다. 이러한 까닭에 이 작업을 위한 경비가 최초 개발할 당시의 경비보다 훨씬 초과할 경우도 있게 된다.

2. 대형 소프트웨어 개발 작업에서의 활동

대형 소프트웨어를 개발하는데 필요한 행동을 설명하기 위하여 국세청에서 실시하는 세금징수 업무를 생각해 보기로 한다. 세금징수 업무는 상당히 방대한 업무이며, 이 업무에 대한 소프트웨어를 개발하는 절차 및 행동은 다음과 같이 요약할 수 있다.

(1) 요구분석(requirement analysis)

우선 먼저 수천 페이지가 넘는 세법(tax law)에 관계되는 문서를 완전히 이해하여야 하며, 문서에 내재되어 있는 애매모호한 부분과 모순이 되는 부분은 완전히 수정하거나 제거되어야 한다. 이 활동을 위해서는 소프트웨어 개발자 뿐만이 아니라 세금 전문가를 포함한 많은 인원이 소요된다.

(2) 설계(design)

이 단계에서는 소프트웨어 시스템의 전반적인 구조를 설계한다. 이 활동을 위하여 위와 마찬가지로 많은 인원이 소요되며 소프트웨어 공학에 관한 방법론(methodology)과 기술이 필요하다. 또한 세법과 소프트웨어 요구 문서를 보다 더 명백하게 하고 세법이 차후에 수정 되었을 때 소프트웨어를 쉽게 수정할 수 있도록 하기 위하여 세금 전문가와 상의 하여야 한다.

(3) 코딩(coding)

소프트웨어 시스템의 여러가지 요소들에 관하여 원시코드를 작성한다. 코더(coder)들의 활동은 각각 독립적으로 수행될 수 있지만, 이들이 소프트웨어 설계자와 세금 전문가들과 상의해야 함은 두말할 필요가 없다.

(4) 통합(integration)

상기에서 작성된 소프트웨어 시스템의 각 요소에 대한 원시코드를 하나의 완전한 시스템으로 통합한다. 이것은 그룹활동이며, 설계자와 코더들은 서로 상의해서 통합한다.

(5) 테스팅과 검증(testing and validation)

완성된 소프트웨어 시스템이 세법을 정확하게 반영하고 있는지를 테스트 한다. 세법의 복잡성에 비추어 이 작업은 상당한 시간이 소모되고 설계자와 코더 그리고 세금 전문가 사이에 상호의 의견이 교환되어야 한다.

(6) 보수 및 개선(maintenance and evoluation)

세법이 변경됨에 따라 시스템은 변경되어야 한다. 세법은 새로운 세법이 제대로 효과가 발휘 되기도 전에 또다시 변경되는 경향이 있다. 사실은, 세금 징수 업무를 취급하는 소프트웨어 시스템이 개발이 완료되어 가동되기도 전에 세법이 변경되기도 한다. 이러한 문제점을 해결하기 위해서 철저한 문서작성과 소프트웨어의 재구성 방법론의 연구가 제시되기도 한다.

이상과 같은 소프트웨어의 활동단계를 라이프 싸이클(life cycle)로 설명한다. 소프트웨어 엔지니어링의 주요관심 역시 라이프 싸이클의 단계별로 최적한 방법론(methodology)과 툴(tool)을 개발하여 소프트웨어 개발의 생산성을 높이고, 품질을 보증하여 향상시킬 수 있도록 연구하는데 있다.

따라서 소프트웨어 엔지니어링은 모든 소프트웨어, 즉 시스템 프로그램, 프로그래밍 언어, 응용프로그램 모두를 대상으로 하고 있으며, 이러한 소프트웨어는 인간이 개발하고 인간이 사용함으로 인공지능에서 다루는 추론이나 탐색방법, rule을 기반으로한 방법론들의 연구까지 이 분야에 포함시키고 있다.

Ⅳ. 소프트웨어 공학을 위한 인공지능의 역할

지난 10여년 동안 인공지능(AI : artificial intelligence)분야의 연구 중점은 알고 있는 지식(knowledge)을 어떻게 표현하고 어떻게 이용할 것이냐? 하는데 집중되었다. 또한 소프트웨어 공학 활동은 지식 집약적(knowledge-intensive)이기 때문에 인공지능 기술을 소프트웨어 공학 활동에 적용하는 방안이 중요한 논제로 부각되고 있다.[2]

앞에서 살펴 본 바와 같이 여러 가지 유형의 지식이 소프트웨어 공학 활동을 위하여 필요하다. 프로그래밍 기술에 관한 지식은 소형 소프트웨어 시스템을 개발하는데 긴요하다. 또한, 소프트웨어 공학에 관한 방법론(methodology)은 대형 소프트웨어 시스템을 개발하는데 필요하다. 사용하는 컴퓨터의 특성과 구조에 대한 지식은 두개의 시스템 모두에게 필요하다.

적용 영역에 관한 지식과 목적 소프트웨어(target software)에 관한 지식은 모든 활동에 공통적으로 적용되는 것은 아니지만 상당한 관심거리가 되고 있다. 적용 영역에 관한 지식은 대형 시스템과 소형 시스템 개발 활동에 포괄적으로 적용된다. 목적 소프트웨어에 관한 지식은 시스템의 보수와 개선을 위하여 특별히 필요하다. 표 1 은 적용업무에 관한 지식과 목적소프트웨어에 관한 지식이 소프트웨어 공학활동을 위하여 공헌하는 역활을 보여주고 있다.

표 1. 소프트웨어 공학 활동에 사용되는 지식

구 분	활 동	관 련 지 식		
		적용 업무	목적 소프트웨어	
소형 시스템	소프트웨어 사양	○	×	
	분 리		×	
	구 현		×	
	테 스 팅	○		○
	최 적 화	○	×	○
	인 준	○		○
	개 선		×	○
대형 시스템	요 구 분 석	○	×	
	설 계	○	×	
	코 딩	○	×	○
	통 합			○
	테스팅, 인 준	○		○
	보 수, 개 선	○		○

범례 : ○ : 관련 지식을 사용하는 활동
× : 차후에 지식을 사용하기 위하여 지식을 창출하는 활동

표 1 을 살펴보면, 적용 업무와 목적 소프트웨어에 관한 많은 량의 지식이 명확하게 기록되는 경우가 드물다는 것을 알 수 있다. 적용 업무에 관한 지식은 소프트웨어 사양과 요구 분석에 적용되지만, 사양과 요구에 대한 배경을 추론하는 요령에 대해서는 거의 기록되지 않는다. 또한 왜 이렇게 구현 하였으며 왜 이렇게 설계 하였는가에 대해서도 통상 서술하지 않는다. 이것들은 결국 소프트웨어 개발자를 중요한 문제에 봉착되게 만들며 또한 소프트웨어 경비를 증가시키는 요인이 된다. 이러한 불확실한 지식을 보다 더 유효 적절하게 관리되고 이용될 수 있도록 하기 위해서는 컴퓨터의 지원이 긴요하다. 컴퓨터 지원은 결국 인공지능 기술에 좌우되며, 소프트웨어 공학 활동을 위하여 적용할 수 있는 인공지능 기술은 다음과 같이 요약할 수 있다.

(1) 경험적인 탐색법(heuristic search paradigm)은 소프트웨어 설계와 구현 활동에 관한 표준 모형(standard model)을 만들기 위하여 사용될 수 있다. 이 기술의 중요한 장점은 적용 가능한 여러 가지의 방법중에서 적용된 방법을 왜 선택하였는지를 정확하게 서술할 수 있다는 점이다.

(2) 적용 업무와 목적 소프트웨어를 분석하고 추론하기 위하여 추론(inference system)이 필요하다. 물론, 이와 같은 시스템은 적용 업무와 목적 소프트웨어에 관한 지식을 저장하기 위하여 사용되는 지식표현 시스템(knowledge representation system)과 연결되어야 한다

(3) 추론 시스템은 수학적인 기술을 구현하기 위하여 필요하다. 이와 같은 기술은 소프트웨어를 구현하거나 최적화(optimization) 하기 위해서 사용되어 질 수 있다.

(4) 소프트웨어 공학을 위한 방법론과 프로그래밍 기술에 대한 전문적인 의견을 표현하고 사용하기 위하여 지식표현 시스템과 규칙 기반 시스템(rule-based system)이 필요하다.

(5) 적용 영역에 관한 지식을 저장하고 검색하기 위하여 지식 표현기술이 필요하다. 이것은 요구사항을 분석하고, 소프트웨어를 정의하며, 개발된 소프트웨어를 개선하는 동안에 특히 필요하다.

(6) 소프트웨어가 개발되어진 역사에 대하여 지식을 저장하고 검색하기 위하여 지식표현 기술이 필요하다. 이 지식은 요구사항 분석, 소프트웨어 사양, 설계, 구현하는 동안에 창출되며 대부분의 다른 활동 동안에 활용되어 진다.

이와 같이 소프트웨어 엔지니어링의 연구는 소프트웨어를 개발하는 모든 프로젝트에 적용되고 있으며 소프트웨어를 하드웨어와 같은 제품(product)개념으로 발전시키려는 연구를 공학적인 측면에서 추진하고 있다.

소프트웨어는 그 물성(物性)을 정의하기 어렵기 때문에 이론적인 전개 역시 까다롭다. 소프트웨어의 개

발요소인 프로그램과 데이타를 모두 데이타 타입으로 정의하고 이들의 영역(domain)을 설정하여 관계를 찾고, 관계에 의해서 정의된 구조의 특성을 연구하고 있다. 프로그램과 데이타를 데이타 타입으로 설명하여 抽象化된 데이타 타입의 어떤 영역에서 연산되는 규칙을 도출할 수가 있으며, 이 영역을 대수적 구조로 설명함으로서 요소들의 관계가 정의될 수 있고, 소프트웨어의 신뢰성에 관한 평가이론들이 정립되어 갈 것이다.

신뢰성 연구와 함께 추상화된 데이타 타입의 이론이 소프트웨어 엔지니어링의 경험적인 연구결과를 바탕아래 소프트웨어 과학으로서의 기반을 굳히며 소프트웨어의 개발 기술에 획기적인 발전을 기대할 수 있을 것이다.

參 考 文 獻

[1] Baker, A.L, et al., "A comparision of measures of control flow complexity", in Proc. 3rd Int. Computer Software & Application Conf., Chicago, IL, 1979.

[2] Barstow, D., "Artificial intelligence and software engineering", in Proc, 9th Int. Conf. Software Eng., Monterey, CA, 1987.

[3] Brooks, F.P., *The mythical man month*, Addison-wesley Publ. Co., Reading, Mass., 1975.

[4] Boehm, B.W., "Software and its impact: a quantitative assement", Datamation, May 1973.

[5] Coulter, N.S., "Software Science and Cognitive Psychology", *IEEE Trans. Software Eng.*, vol.. SE-9, Mar. 1983.

[6] Douglas, T.R., et al., "Software engineering: Process, Principles, and goals", Computer, May 1975.

[7] "Everything about the narrows bridge is big, bigger or biggest", Eng. News Record 166, June 29, 1961.

[8] "Narrows bridge opens to traffic", Eng. News Record 173, Nov. 19, 1964.

[9] Gorden., R.D., "Measuring improvement in program clarity", *IEEE Trans. Software Eng.*, vol. SE-5, Mar. 1979.

[10] Halstead, M.H., *Elements of Software Science*, New York, Elesevier 1977.

[11] Halstead, M.H., "Guest editorial on software science", *IEEE Trans. Software Eng*, vol. SE-5 Mar. 1979.

[12] Kron, H., and DeRemer, F., "Programming in the large versus programming in the small", *IEEE Trans. Software Eng.*, Jan. 1975.

[13] Ottenstein, L.M., "Quantitative estimates of debugging requirements", *IEEE Trans. Software Eng.*, vol. SE-5 Sept. 1979.

[14] Woodfield, S.N., "An experiment in unit increase in problem Complexity", *IEEE Trans. Software Eng.*, vol. SE-5, Mar. 1979.

[15] Zelkowitz, M.U., "Perspective on software engineering", *Computing Surveys*, vol. 10, no. 2, June 1978. *

♣ 用 語 解 說 ♣

Application Study (응용 연구)

정해진 기능이나 작동을 하기에 알맞는 컴퓨터 사용 체제를 결정하며, 특정한 목적에 적합한 장비 선택의 기초를 설정하는 과정

Archiving (기록 보관저장)

컴퓨터 시스템에서 기록 보관소 내에 화일을 적재하고 이를 관리하는 작업. 이 작업은 운영 요원지시에 따라 운영체제에 의해 수행된다

Arithmetic Expression (산술식)

데이타의 이름, 숫자, 정수명들과 산술 연산자들을 조합하여 하나의 수치로 계산되어질 수 있도록 표현한 수식