소프트웨어 생산성의 명암

한국과학기술원 권용래*

1. 서 론

소프트웨어 공학에 대한 관심은 미국 항공우주국의 우주선 및 각종 인공위성, 국방성의 전략무기 시스템, IBM사의 메인프레임 컴퓨터 시스템 개발 등 초대형 프로젝트를 추진하는 과정에서 비롯되었다고 볼 수 있다. 프로젝트의 중요성이 워낙 지대하여 개발에 소요되는 예산이나 인원을 크게 걱정하지 않아도 되는 분위기에서 소프트웨어 엔지니어들은 당시의 소프트웨어 기술로서는 도저히 감당해 내기 어려운 대규모이며 복잡성이 큰 시스템을 다루어야 했고 당연히 과거에는 경험하지 못한 참담한 실패에 직면하였다 과거의 모든 공학 분야가 그러했듯이 소프트웨어 공학도 60년대말로부터 시작된 소프트웨어 응용의 급속한 확장의 결과로 초래된 각종 문제에 대응하는 노력의 과정에서 탄생하였다

이와 같이 연륜이 짧은 소프트웨어 공학은 형편없이 낮은 생산성과 품질에 시달여야 했고 메모리 소자, 프로세서, 컴퓨터 시스템 및 주변 기기의 눈부신 발전에 대비되어 늘 비판의 대상이 되곤 하였다. 마침내 이러한 상황을 소프트웨어 위기로 규정하였고 그 중요성을 고려하여 국가가 개입하여 탈출구를 마련하고자 하기도 하였다. 한 해 전 세계의 소프트웨어 수요가 8000억 달러에 달한다고 한다. 따라서 소프트웨어 생산성을 10%만 제고 한다고 해도 매년 800억 달러의 경제적 가치가 있는 셈이다. 그러므로 소프트웨어 생산성을 높이는 일은 기술적 과제에 그치지 않고 세계 각국의 경제적인 관심사로 대두되고 있다.

그러나 소프트웨어 공학이 수수방관만을 한 것은 아니다. 꾸준히 새로운 개발 기법을 연구하고 그러한 기법을 도구의 형태로 구현하여 상품화를 시도하고 있으며 이미 개발되어 있는 소프트웨어의 상당 부분을 재사용함으로써 생산성의 획기적 제고를 시도하고 있다 또한 부품 조립 방식에의한 소프트웨어 생산 방식이 대두되어 소프트웨어 공학의 새로운 패러다임으로서 21세기의 핵심 기술로서까지 주목을 받고 있다[1]. 그러나 아직도 소프트웨어 생산은 인력 집약적 양태를 벗어나고 있지 못하고 있으며 당분간 소프트웨어 생산성의 비약적 발전은 기대하기 어려울 것으로 보인다.

2. 소프트웨어 생산성

프로그래밍이 야기하는 문제점들을 해결하려는 노력의 산물로 구조적 프로그래밍 기법[2]이 등장하였다. 프로그래밍 기술이 어느 정도 궤도에 오르자 설계 기술의 중요성이 인식되었다. 구조적 설계 기법[3]은 이러한 인식의 소산이었다. 곧 이어 많은 설계상의 문제점은 요구 사항을 잘못 이해하는 데서 비롯된다는 사실을 발견하였다. 결국 소프트웨어 개발 과정에서 당면하는 문제를 하나 하나씩 해결해 나가는 과정에서 소프트웨어 전 개발 공정을 뒷받침하는 개발 기술을 확보되었다. 이 때 비로서 소프트웨어 공학은 공학으로서의 면모를 갖추게 되었고 독립적인 공학 분야로 자리잡기 시작하였다. 그러나 이러한 개발 기술은 공정별 작업 수순을 체계화 하여 품질 개선에 크게 기여하였지만 새로운 기법이 반드시 생

* 종신회원

산성 향상으로 이어졌다고 보기는 어렵다. 놀라운 생산성 향상을 보여준다고 주장하는 일부 기법들도 그 사실이 현장에서 실험을 통하여 입증된 경우는 거의 없었고 기법을 고안해낸 사람들의 희망적 견해로 판명되는 일이 많았다. 소프트웨어 공학은 현장에서 입증된 기법들의 축적한 다음, 차후 개발에 당연한 기술로서 활용하는 데 부진했다. 끊임없이 새로운 기법들이 새로운 기법들이 소개되고 기대에 부응하지 못하면 폐기되곤 하였다.

소프트웨어 생산성은 보통 단위 시간당 완성하는 프로그램 문장의 수로 측정된다. 생산성을 보다 정밀하게 표현할 수 있는 많은 연구가 있어 왔지만, 생산성을 정확하게 나타내지 못한다는 지적에도 불구하고 여전히 단위 시간당 완성되는 프로그램 문장수가 생산성의 척도로서 널리 쓰이고 있다. 생산성을 논의하는 데에는 일정한 품질 수준을 유지한다는 전제가 필요하다. 품질을 고려하지 않는다면 생산성은 고무줄처럼 늘일 수 있기 때문에 생산성을 경제적 가치와 연계하는 의미가 없어진다. 생산성을 개선하려고 하기 전에 우선 현재의 생산성을 알 수 있어야 한다. 그에 따라 현재의 생산성으로도 충분한지 아니면 현저한 개선이 필요한지 여부를 결정할 수 있고 그에 따라 개선책을 내놓을 수도 있다.

3. 생산성 개선의 가능성

B. Boehm은 소프트웨어 생산성을 제고할 수 있는 가능성으로 1) 훌륭한 인력의 확보, 2) 개발 작업의 효율화, 3) 불필요한 개발 작업의 제거, 4) 재작업의 제거, 5) 보다 단순한 제품의 제작, 6) 부품의 재사용 등을 열거하였다[4]. 이처럼 생산성을 생산하는 데에는 기술적 부분과 기술 외적 부분이 있다. 기술 외적인 요소로는 크게 개발 인력의 자질과 프로젝트 관리 기술을 들 수 있다 소프트웨어는 노동집약적 방식으로 개발되기 때문에 생산성의 인력에 대한 의존도는 절대적이다. 따라서 대학이 현장에서 사용 가능한 기술을 교육시키는 일이 중요하고 대학의 기술이 효과적으로 산업체에 이전될 수 있어야 한다. 그리고 대규모 소프트웨어 개발에는 많은 인력이 동원되어 장기간 작업을 해야 한다. 그러므로 효과적인 프로젝트 관리는 개발 기술 못지않게 중요하다. 실제로 그사이의 생산성 개선에는 다른 분야가 발전시킨 프로젝트 관리 기술을 소프트웨어 개발에 적용시켜 얻은 것이 큰 몫을 차지한다고 볼 수 있다. 가능성 2)에서 5)까지는 개발 기술의 개선에 해당한다. 지난 30여년 동안에 열거할 수 없을 정도로 많은 기법들이 소개되어 왔다. 대부분은 학술적인 연구 결과이고 개발 현장에서 효과가 입증된 것은 드물다. 소프트웨어에 수반하는 각종 문서를 자동적으로 산출함으로써 인력을 줄이는 일, 소프트웨어 도구를 사용하여 엔지니어의 부담을 더는 일, 공정을 개선하는 일 등이 이 범주에 속하는 노력이다. 가능성 6)은 발상의 전환이다. 즉 이미 개발되어 있는 소프트웨어를 새로 개발해야 하는 소프트웨어의 일부로 사용함으로써 새로운 개발을 가급적 줄이고자 하는 것이다. 다음은 기술적인 면에서 생산성 제고에 기여할 것으로 기대되고 있는 수단들이다.

소프트웨어 개발 방법론과 개발 환경 - 이제까지 고안된 개발 기술 중에서 생산성 향상에 크게 기여한 것으로는 요구사항 모델링 기법을 꼽을 수 있다. 데이터 흐름도를 중심으로 한 구조적 분석 기법[5]이 널리 보급되어 큰 성공을 거두었고 최근에는 이것이 객체 지향 모델링[6, 7]으로 대체되어 가고 있는 중이다. 일단 소프트웨어 전공정을 커버하는 기술이 확보되자 개별적인 기술을 구현하는 도구를 고안하여 인력을 대체하려고 노력하였다. 이러한 노력을 컴퓨터 지원을 받는 소프트웨어 공학(CASE: Computer-Aided Software Engineering)이라고 부른다[8]. 생산성의 획기적인 향상은 전 공정에서 자동화 도구 즉 CASE 도구를 사용할 수 되어 있을 때라야 가능할 것이다. 그러나 소프트웨어 개발은 아직도 일부 개발 단계에서 상품화되어 있는 CASE 도구가 사용되고 있을 뿐 대체로 보아서는 원시 수공업적인 제작 방식을 벗어나지 못하고 있다 다른 공학에서 볼 수 있는 전공정 자동화는 당분간 기대하기 어려울 것이다. 따라서 개발 속도의 향상을 통한 생산성 제고는 한계가 있다.

소프트웨어 개발 환경은 개발방법론과 소프트웨어 도구, 개발에 사용되는 하드웨어 시스템, 개

발자의 작업 환경을 고려한 통합된 생산 체제를 구성하고자 하는 시도이다[9]. 개념적인 당위성과 국가적 지원에도 불구하고 기술적 난관, 구축에 소요되는 방대한 노력과 구축으로 얻어질 경제적 효과에 대한 회의 때문에 현실화되지 못했다고 볼 수 있다. 단 프로그래밍 개발 환경은 소프트웨어 개발 환경에 비하여 구축이 용이하고 경제적 효과가 인정되어 이제 거의 프로그래밍 언어와 구분 없이 사용하기에 이르렀다. Smalltalk, Visual C++, Java 등은 모두 프로그래밍 언어 자체와 언어를 지원하는 에디터, 디버거 등을 포함하여 말한다

재사용과 컴포넌트 소프트웨어 - 이미 개발되어 있는 소프트웨어의 일부를 다른 소프트웨어 시스템에 재활용 한다면 새로이 개발하지 않아도 되어 노력이 크게 절감됨은 물론 그 동안의 사용을 통하여 품질을 어느 정도 되어 있다는 부수적인 이익이 있다. 이것이 소프트웨어 재사용(Reuse) 개념이다[10]. 재사용이 효과적이기 위해서는 재사용할만한 소프트웨어들을 간단한 노력으로도 확보할 수 있어야 한다. 그러려면 충분한 소프트웨어 부품들이 체계적으로 분류되어 있고 간단한 검색을 통하여 찾아낼 수 있어야 한다. 자기가 필요한 소프트웨어 부품을 충분히 확보하는 데는 엄청난 초기 투자가 필요하기 때문에 경제적 이익을 기대하기 어렵다.

규격화되고 검증된 소프트웨어 부품들의 획득과 조립 방식으로 소프트웨어를 개발하고자 하는 다른 분야에서는 보편화되어 있는 방식이지만 소프트웨어의 경우에는 일대 패러다임의 변화라 할 수 있다. 이와 같은 컴포넌트 기반의 소프트웨어 공학(CBSE: Component-Based Software Engineering)은 다양한 이점을 제공해 줄 수 있는 새로운 기술로서 선진국에서도 이것을 21세기의 핵심 기반 기술로 꼽고 있다[11, 12]. 컴포넌트 조립 방식의 소프트웨어 개발이 활성화 되기 위해서는 필수적으로 다양하고 품질 높은 소프트웨어 컴포넌트 시장이 형성되어야 한다. 그래야 소프트웨어 수요자는 주어진 비용의 범위 내에서 적절한 부품을 구입하여 의도한 시스템을 조립할 수 있을 것이며 부품의 수요가 충분해야 질 좋은 부품(COTS: Commercial Off-the-Shelf)을 개발하여 시장에 내 놓을 경제적 동기가 유발될 것이다. 일단 컴포넌트 기술에 대한 경제적 동기가 부여되면 소프트웨어 부품 산업이 자연적으로 성장할 것이고 상당 규모의 소프트웨어 시장이 형성되어 소프트웨어 수요자들이 편리하게 그리고 경제적으로 이용할 수 있어 전반적으로 한 차원 높은 생산성이 달성될 수 있을 것이다

4. 결 론

고작 30년에 지나지 않는 일천한 역사에도 불구하고 소프트웨어 공학은 개발 기술면에서 괄목할 진전을 이루었고 어느 정도의 품질 수준을 유지할 수 있게 되었다. 그러나 여전히 소프트웨어의 낮은 생산성과 품질은 비판의 대상이 되고 있으며 소프트웨어가 정보화 사회를 실현하는 데 걸림돌이 되고 있다는 지적을 받고 있다. 그러나 개발 대상 문제의 다양성, 하드웨어 시스템의 꾸준하고도 급속한 변화, 예를 찾아 보기 어려운 새로운 형태의 소프트웨어에 대한 끊임없고 조급한 수요 때문에 체계적인 생산 기술을 확보하기 어렵다는 점을 이해할 필요가 있다. 당분간은 제대로 작동하는 소프트웨어 시스템을 공급하는 것이 우선적으로 고려되어야 하고 생산성을 제고하는 것은 부차적인 고려 사항이 될 듯하다. 그러나 소프트웨어 공학은 새로운 기술을 추구하는 한편, 이미 확보된 기술과 도구를 적절히 패키징하여 언제나 증가해 가기만 하는 수요에 적절히 대체해야 할 것이다.

참고문헌

[1] Information Technology for the Twenty-First Century: A Bold Investment in Americas Future, Presidents Information Technologies Advisory Committee. U. S. A. Jan. 1999.

[2] H. D. Mills, Structured Programming: Retrospect and Prospect, IEEE Software. Nov. 1986.

[3] W. Stevens, G. Myers and L. Constantine, Structured Design, IBM Systems Journal, May 1974.

[4] B W. Boehm, Improving Software Productivity, IEEE Computer, Nov. 1987.
[5] D. T. Ross, Structured Analysis(SA): A Language for Communicating Ideas, IEEE Trans. On Software Engineering, Jan. 1977.
[6] I Jacobson, Object-Oriented Software Engineering, Addison-Wesley, 1993.
[7] J. Rumbauch, et al., Object-Oriented Modeling and Design, Prentice Hall, 1991.
[8] C. Gane, Computer-Aided Software Engineering, Prentice Hall, 1990.
[9] L. Osterweil, Software Environment Research: Directions for the Next Five Years, IEEE Computer, Apr. 1981.
[10] P. A V. Hall, Software Components and Reuse Getting More Out of Your Code, Information and Software Technology, Feb. 1987, Butterworth Scientific, Inc.
[11] C. Szypersky, Component Software: Beyond Object-Oriented Programming, Addison-Wesley, 1998.
[12] N. Szyperski, Component Software: A Market on the Verge of Success, The Oberon Tribune, 2(1), Jan. 1997.

권 용 래

1969 서울대학교 문리대학 이학사(물리학)
1971 서울내학교 대학원 이학석사(물리학)
1971.9~1971.9 육군사관학교 전임강사
1978.12 미국 피츠버그대학 이학박사(물리학)
1978.8~1983.7 미국 Computer Sciences Corporation 연구원
1983.8~현재 한국과학기술원 전산학과 교수
관심분야:Software Specification Development Tools, Programming Environment, Human-Computer Interface, Computer Supported Cooperative Works CASE
E-mail:kwon@cs.kaist.ac.kr

'99 주요행사 연간일정표

월 별	주 요 행 사		
	행 사 명	일 시	장 소
3월	Y2K해결을 위한 호남지역 워크숍	4일(목) 14:00	광주대학교
4월	Y2K해결을 위한 충청지부 워크숍	16일(금) 14:00	한남대학교
	제26회 임시총회 및 춘계학술발표회	23일(금)~24일(토)	목포대학교
5월	Y2K 해결을 위한 영남지역 워크숍	28일(금) 10:00~12:30	부산대학교
6월	제12회 정보문화의 달 기념 행사	3일(목) 14:00	전북대학교
	Y2K 문제 해결을 위한 심포지움	29일(화) 14:00	과학기술회관
7월	제1차 리눅스 포럼	27일(화)~28일(수)	서울 롯데호텔
10월	제26회 정기총회 추계학술발표회	22일(금)~23일(토)	광운대학교
11월			
12월	제17회 정보산업리뷰 심포지움	10일(금) 13:00	코 엑 스
	1999년도 학회 송년회	10일(금) 18:30	무역크럽