소프트웨어 프로젝트 평가모델을 통한 소프트웨어 메트릭스 분석

An Analysis of Software Metrics Using the SPEM(Software Project Estimation Model)

이재기(J.K. Lee) ACE시스템팀 책임연구원
신상권(S.K. Shin) ACE시스템팀 연구원
남상식(S.S. Nam) ACE시스템팀 책임연구원, 팀장
박권철(K.C, Park) 네트워크전략연구부 책임연구원, 부장

본 논문은 대형 프로젝트를 수행하는 데 있어서 필요한 리소스, 인력, 개발비용 및 소프트웨어 소스에 대한 데이터를 추정하여 프로젝트의 효율성을 평가하는 모델인 소프트웨어 프로젝트 평가모델을 이용하여 기 수행된 프로젝트의 경험데이터와 수행되고 있는 프로젝트의 소프트웨어 메트릭스(metrics) 데이터를 활용하여 생산성, 품질, 자원투입 효과, 개발될 소프트웨어 소스 규모 등을 추정해 보고 이를 경험적인 모델(empirical model)에 적용하여 프로젝트 별로 평가, 비교 분석해 본다. 또 향후 유사 프로젝트 관리(similar project management)에 필요한 사항들을 제안한다.

I. 서 론

초기 소프트웨어 가격은 시스템 전체에서 차지하는 비율이 매우 적었기 때문에 발견되는 에러에 대한 소프트웨어 해결 비용이 매우 미약했다. 그러나 현재는 시스템에서 소프트웨어가 차지하는 가격이 매우 높기 때문에 개발비용 및 효과에 대한 추정 방법에 많은 관심을 기울이게 되었다. 이러한 관심 속에 대두된 것이 소프트웨어의 개발규모나 기능에 대한 복잡도(complexity)를 가지고 평가하는 소프트웨어 생산성 모델(software productivity model)이다. 다시 말해서 프로그램의 LOC(Line of Code)나 FP(Function Point) 수를 추정하여 소프트웨어의 생산성을 평가하는 프로젝트 평가 모델(Project Estimation Model: PEM)에 관심을 가지게 되었다.

이 모델은 각각의 소프트웨어 요소들에 대한 "size"나 "variable"에 대한 생산성 측정(productivity metrics)으로 때로는 과거의 프로젝트 수행 결과로부터 얻은 정보를 이용하기도 한다. 다시 말해서 cost data, size-oriented data, function-oriented data 등을 활용하여 소프트웨어의 품질(quality)이나 생산성(productivity)에 대한 평가를 하는 모델이다[1].

II. 소프트웨어 프로젝트 관리

소프트웨어 프로젝트 관리를 성공적으로 수행하기 위해서는 처리할 일의 양이나 자원의 관리 또는 milestone에 대한 추적 관리, 주어진 일정 관리 및 effort(cost) 분석 등이 수반된다. 이를 성공적으로 이끌기 위해서는 우선 소프트웨어 프로젝트 계획에 대한 연구(research)와 추정(estimation)이 뒤따라야

한다. 즉, 소프트웨어 각 요소에 대한 범위(scope)가 정의되고 이에 대한 연구가 행해져야 한다.

그밖에 리소스 분석, 생산성 및 품질에 대한 소프트웨어 메트릭스의 분석, 소프트웨어 프로젝트 추정 및 분석 기술을 이용한 프로젝트 추정, 경험적인 데이터를 활용한 프로젝트 분석과 자동화된 툴에 의한 추정 등이 수행되어야 한다. 특히 양질의 소프트웨어를 획득하기 위한 PERT(Program Evaluation and Review), CPM(Critical Path Method)을 도입한 프로젝트 스케줄링 관리 등과 소프트웨어 조직에 대한 분석 등을 통해 프로젝트 계획의 실천이 수행되어야 한다. 다시 말해서 소프트웨어 관리상의 문제로서 아래와 같은 사항이 제기된다.

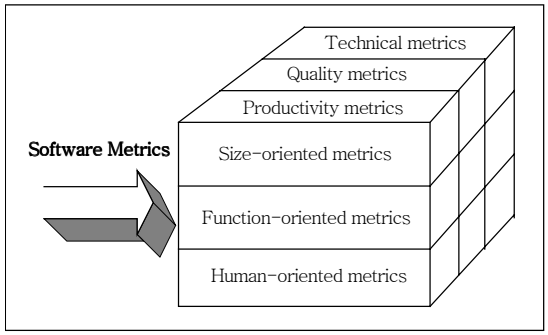
① scope and resource, technical staff와 customer 간의 대화
② 비용와 스케줄 관리에 대한 리뷰
③ 프로젝트 전체에 대해 참여자에게 문서를 통한 소프트웨어 개발에 대한 접근방법 제공 등이다.

III. 소프트웨어 측정

소프트웨어 측정(software metrics) 방법은 크게 2가지 방법으로 나뉘는데 첫째 직접측정(direct measure)으로 하나의 묶음으로 소프트웨어의 크기로, 둘째는 단위 생산(unit product)된 또는 거부된 소프트웨어의 품질 등으로 측정이 되는 간접측정(indirect measure)으로 나눈다.

- 직접측정
  cost, speed(delivery), effort, LOC, memory size, errors 등을 측정
- 간접측정
  function, quality, complexity, reliability, maintainability, efficiency 등을 측정

위와 같이 소프트웨어 측정에 대한 범주를 정리해보면 (그림 1)과 같다.



(그림 1) 측정 범주(metrics categorization)

- Quality metrics
  사용자 요구사항에 대한 소프트웨어의 품질척도를 explicit & implicit 하게 표시
- Productivity metrics
  소프트웨어 엔지니어링 절차에 대한 출력에 초점을 둠
- Technical metrics
  논리적 복잡도나 모듈화 정도 등 소프트웨어의 특성에 초점을 둠
- Size-oriented metrics
  소프트웨어 엔지니어링 절차에 따른 산출물을 직접 측정하여 수집함
- Human-oriented metrics
  개발자의 태도나 툴 또는 방법 등에 대한 효율성에 대한 정보 수집
- Function-oriented metrics
  간접측정에 의한 정보로 품질, 신뢰도, 복잡도, 유지보수성 등이 속한다[2],[3].

1. Size-Oriented Metrics

개발 절차나 소프트웨어에 대해 직접 측정하는 방법으로 생산성, 품질, 비용, 문서화에 대해 (1)~(4)로 표시할 수 있다.

① 생산성(productivity) = KLOC/person-month ---------- (1)
② 품질(quality) = error/KLOC ---------- (2)
③ 비용(cost) = $/KLOC ---------- (3)

④ 문서화(documentation) = pages document/ KLOC ----------(4)

2. Function-Oriented Metrics

프로그램의 기능성(functionality)이나 유틸리티(utility)에 대한 측정방법으로 1979년 Albrecht에 의해 제안된 방법으로 프로그램의 복잡도나 소프트웨어의 information domain에 대한 측정 등을 카운트하는 경험적인 방법(empirical study)이 도입된다. 즉, 입·출력 수, 파일 개수, 외부 인터페이스 수, 사용자 질의 수 등이 관련되며, 기능 수(FP)는 아래와 같은 관계식으로 표현된다. 그 후 1983년에 Albrecht and Gaffney[AG83]에 의해서 수정, 제안되었다.

$FP = count.total \times [0.65 + 0.01 \times SUM(F_i)]$

F_i: complexity adjustment value(i = 1 to 14)

이 모델의 생산성, 품질, 비용, 문서화에 대한 추정식은 (5)~(8)로 표현된다.

① 생산성(productivity) = FP/person-month ---------- (5)
② 품질(quality) = error/FP ---------- (6)
③ 비용(cost) = $/FP ---------- (7)
④ 문서화(documentation) = pages document/ FP ---------- (8)

그 밖에 언어별 FP에 대한 LOC 규모는 <표 1>과 같다.

<표 1> 언어별 FP에 대한 LOC 규모

언어(language)	LOC/FP
COBOL	110
PL/1	65
4GL	25

3. LOC and FP Estimation

소프트웨어 프로젝트를 추정하는 방법은 소프트웨어 규모(혹은 크기)나 과거에 수행한 프로젝트로부터 얻은 baseline 메트릭스로서 개발비용이나 효과에 대한 변수들을 추정하여 적용하는 방법이 있다. 즉, FP에 의한 간접 추정시 14개의 복잡도 조정값들(complexity adjustment values)을 적용하는데 이와 같이 예상이 되는 추정 값들을 조정하여(weighted) 환경 변수에 따라 최적(optimistic), 보통(most likely), 열악(pessimistic)으로 추정하는 프로젝트 수행효과(E)는 아래와 같이 표현된다.

$E = (a + 4m + p)/6$

a: optimistic, m: most likely, p: pessimistic

즉, LOC 또는 FP로 표현되는 추정자(estimation factor)로서[4] 프로젝트에서 개발된 소프트웨어 규모가 130만 라인이라 하면 Esterling에 의해 1980년도에 제안된 time-study model에서 추출된 <표 2>의 파라미터를 적용해 보면

$E = (0.05 + 0.10 \times 4 + 0.15)/6 = 0.1$ KLOC/PY

PY: Person-Year

가 된다. 다시 말해서 ACE2000 시스템인 경우 실제 프로젝트에 참여한 프로그래머가 20명, 총 소프트웨어 규모(LOC)가 130만 라인으로 개발자 1명이 한 달에 약 440 LOC 정도의 프로그램을 개발한 셈이 된다.

<표 2> Values for Esterling model parameters

Parameter	Range	Workers	Programmers		
			Optimistic	Typical	Pessimistic
a	0~0.5	-	0.05	0.10	0.15

4. Empirical Estimation Models

이 모델은 프로젝트 단계별(또는 개발순기: project life cycle)로 요구되는 소프트웨어 예측 데이터를 사용하여 추정하는 모델이다[5]-[8]. 이때 적용되는 경험 데이터는 단일 프로젝트(single project) 수행시 얻어진 데이터로 한정될 수 밖에 없어

정확한 데이터 추정에 어려움이 많지만 개발환경이나 소프트웨어의 모든 클래스에 적용함으로써 프로젝트 수행에 필요한 판단 근거를 제시하는 데 활용이 많은 모델이다.

여기에 속하는 대표적인 모델로는 자원평가모델(resource estimation model), 시간추이모델(time study model), COCOMO 모델(Constructive COst MOdel), Putnam 모델 등이 있다.

가. 리소스 모델(Resource Model: RM)

하나 또는 그 이상의 수식으로 표현되는 모델로서 예측 자원투입 효과(predict effort), 개발기간(project duration), 타 프로젝트의 예측 데이터(경험 데이터) 등을 토대로 single-variable, static-multivariable, dynamic-multivariable model 등으로 구분되며, 1980년 Basili에 의해 제안되어 다양하게 평가되는 모델이다.

- Single 변수 모델 식

$R = C_1 \times (\text{estimated characteristic})^{C_2}$

C_1, C_2 : 과거 프로젝트로부터 얻은 경험 데이터

estimated characteristic: effort(E), project duration(D), staff size(S), software documentation(DOC), etc.

나. Walston & Felix[WAL77] 모델

이 모델은 60여 개의 프로젝트로부터(소스 규모는 4,000~467,000라인, 참여자 규모는 12~11,758 PM 정도) 수행된 결과를 토대로 설정한 평가 모델로 여기서 추출된 데이터는 (9)~(13)으로 표현된다.

$$E = 5.2 \times L^{0.91} \quad \text{----------(9)}$$
$$D = 4.1 \times L^{0.36} \quad \text{----------(10)}$$
$$D = 2.47 \times E^{0.35} \quad \text{----------(11)}$$
$$S = 0.54 \times E^{0.06} \quad \text{----------(12)}$$
$$DOC = 49 \times L^{1.01} \quad \text{----------(13)}$$

E : effort(person-months : 자원투입량)
DOC : pages of document(문서작성분량)
L : source lines(KLOC : 소스규모)
D : duration(calendar months : 프로젝트 기간)

다. 비용평가모델의 COCOMO 모델

Boehm에 의해 1982년에 제안된 cost estimation 모델로 소프트웨어의 비용 또는 개발자원투입량(development effort)을 평가해 보는 대표적인 비용평가모델이다.

이 모델은 basic, intermediate, advanced model 등 3가지로 구분 제안하고 있으며, 다양한 프로젝트 성격에 따라서 organic(소형 프로젝트에 적합), semi-detached(중형 프로젝트에 적합하며, 데이터베이스시스템을 사용), embedded mode(대형 프로젝트에 적합하며, 자체 개발환경을 가지고 수행)로 클래스를 달리하고(<표 3> 참조) 15개의 "cost driver" 를 사용하여 6단계의 범위(scale)로 EAF(Effort Adjustment Factor)를 두어 평가한다. 기본 모델식은 (14)~(16)과 같다.

$$E = a_b(KLOC)\exp(b_b) \quad \text{----------(14)}$$
$$D = c_b(E)\exp(d_b) \quad \text{----------(15)}$$
$$E = a_i(KLOC)\exp(b_i)EAF \quad \text{----------(16)}$$

E : effort-person month, D : project duration, EAF : range(0.9~1.4)

<표 3> Intermediate COCOMO 모델

Software project	a_i	b_i
Organic	3.2	1.05
Semi-detached	3.0	1.12
embedded	2.8	1.20

- COCOMO-II PA(Post-Architecture) 모델

개발된 소스를 가지고 규모요인(scaling driver)과 가격요인(cost driver) 등을 이용하여 프로젝트의 생산성, 가격 등을 결정하는 모델이다.

$$E(=MM) = A \times (KLOC)^{B} \times \prod_{i}^{17} EM_i$$

EM_i : effort multipliers of cost driver
B : multipliers scaling driver
A : constant

즉, 소프트웨어 개발에 필요한 소요 공수(E: Effort in Man-Months, 또는 MM으로 표기)의 예측 및 비용 산정을 목적으로 1981년 Dr. Barry Boehm에 의해 최초의 COCOMO가 제안 되었으며, 현대의 발전된 정보화 기술을 반영하여 1995년 수정된 모델인 COCOMO-II 모델이 소개되었다. COCOMO-II 모델은 시스템 개발 순기 중 각 단계별로 다음과 같이 세 가지 모델을 적용할 수 있다.

- Application Composition 모델
  개발 초기 프로토 타입 시제품 개발 시 적용할 수 있는 모델
- Early Design 모델
  개발될 제품의 규모를 알기 어려운 프로젝트의 초기 단계에 기능점수(function point)와 같은 사이징 메트릭을 이용하는 모델
- PA(Post-Architecture) 모델
  프로젝트 개발이 완료되어 소프트웨어 규모 및 가격요인들을 정확하게 적용할 수 있는 모델

또 프로젝트의 소프트웨어 개발 규모를 알고 있는 경우는 PA 모델 적용이 가능하며, PA 모델에서 고려할 변수들은 다음과 같다.

- Total source line 수(SLOC)
- 규모요인
  Precedent, Flexibility, Process Maturity, Architecture/Risk Resolution, Team Cohesion
- 가격요인

- Product Factors:
  Required Software Reliability, Database Size, Product Complexity, Required Reusability, Documentation Match to Life-cycle Needs
- Platform Factors :
  Execution Time Constraint, Main Storage Constraint, Platform Volatility
- Personnel Factors:
  Analyst Capability, Programmer Capability, Application Experience, Programmer Experience, Language and Tool Experience, Personal Continuity
- Project Factors :
  Use of Software Tools, Multi-state Development, Required Development Schedule

그 외에 인건비 산출은 아래의 식으로 표현된다.

직접인건비 = 소요공수(MM)×기술자의 월 급여

위에서 소개한 변수들 외에 요구 사항의 변경으로 일부 코딩되었던 부분을 수정해야 하는 경우 Breakage 변수를 고려할 수 있으며, 기존의 소프트웨어를 수정해서 재사용할 경우 기존 소프트웨어의 수정된 비율(percent design modified, percent code modified, percent of integration required), 소프트웨어의 이해 정도(software understanding) 등의 변수들이 고려된다.

라. Putnam estimation model

이 모델은 1978년도에 Putnam에 의해 제안되어 동적 다 변수 모델(dynamic multivariable model)로서 전체 프로젝트 수행기간 중 각 분야별(요구사항분석 및 설계, 기능구현, 시험, 문서화 etc.) effort 분포를 추정할 수 있는 모델이며, 대형 프로젝트에

<표 4> State of Technology Constant(C_k)

C_k	Software development environment
~ 2,000	Poor
~ 8,000	Good
~ 11,000	Excellent

투입되는 인력 분포의 특징을 알 수 있는 유용한 모델이다. 특히 인력(manpower) 분포 곡선이 Rayleigh 분포를 나타내는 것이 특징이다.

이 모델의 추정식은 아래와 같이 표현되며, 개발환경 지수는 <표 4>에 언급된 값이 적용된다.

$L = C_k K^{1/3} T_d^{4/3}$, $K = L^3 / C_k T_d^4$

L : source line

C_k : state of technology constant(2,000~11,000)

K : effort expended(person-years) over the entire life cycle for software development & maintenance

T_d: project 수행 기간(year)

마. Time study model

Esterling[EST 80]에 의해 제안된 생산성 모델로 개발환경 특성을 적용한 모델이다. 특히 이 모델은 회의 등 비생산적인 활동 변수와 근무시간[1일 8시간 기준]외 초과근무에 대한 변수들을 고려하였다. 변수들에 대한 상세 내역은 <표 5>에 나타내었다.

$w = 0.125[8-8a+g-4r/60p(t+r)/60-k(n-1)(t+r)/60]$ ----------(17)

$T = 7/5nw$ ----------(18)

$c = ns[gd + 8(1+i)]$----------(19)

$E = nw/c$ ----------(20)

$C = c/nw$ ----------(21)

w : useful working time per work day-per person

T : ratio of calendar time to person days to complete project

c : labor cost per work day for an average base salarys

E : cost efficiency

C : project cost per person day

n : project 참여 인원

g : over-time working day/day

<표 5> Values for Esterling Model Parameters [EST 80]

Parameter	Range	Workers	Programmers		
			Optimistic	Typical	Pessimistic
a	0~0.5	-	0.05	0.10	0.15
t	1~20	3	3	5	10
r	5~10	0.5	0.5	2.0	8.0
k	1~10	1	2	3	4
p	1~10	1	1	4	10
i	1~3	0.2	0.2	0.5	1.0
d	1~2	1.5	1.0	1.0	1.5

a : average fraction of workday(운용 또는 간접활동에 소비하는 시간)

t : average duration of work interruptions (minute)

r : average recovery time after interruption

k : number of interruptions/workday(프로젝트에 직접 참여하는 사람)

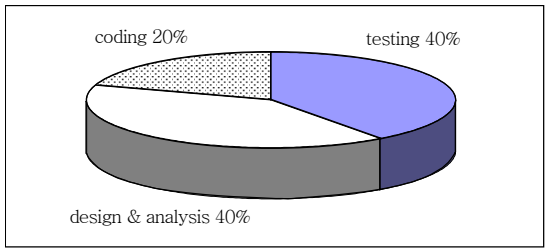
p : number of interruptions/workday(다른 원인으로부터 영향 받는 경우)

i : indirect cost/person(기본연봉 기준)

d : differential pay for overtime(기본연봉 기준)

5. 프로젝트 수행에 소요되는 자원 투입(effort) 분포 현황

일반적으로 각각의 소프트웨어 프로젝트 추정법에 의거하여 effort 분포를 살펴보면 (그림 2)와 같은 분포를 이루는 것으로 밝혀졌다[1]. 즉, 소프트웨어를 설계(분석), 구현, 시험하는 데 40/20/40%가 소요되는 것으로 보고되었으며, 대부분의 소프트웨어 프로젝트 평가모델 분석 결과도 이와 유사하게



(그림 2) 자원투입의 분포

분석되어 발표되고 있다.

IV. 프로젝트 적용 결과

IV장에서는 과거에 경험한 데이터를 참고로 신규 프로젝트에 대한 개발비용, 에러 수, 생산성 규모, 기술문서 작성 등 문서화율에 대한 자료를 가지고 프로젝트 평가모델에 적용한 결과를 추정하고 각 프로젝트(ACE64/256/2000 시스템) 별로 데이터를 비교, 분석해 본다. 그 대상은 아래와 같다.

(데이터 분석 대상)
① 개발비용 평가
② 에러 수
③ 생산성 규모(KLOC)
④ 기술문서 종류 및 분량

1. 문서 작성 현황과 소프트웨어 메트릭스

기술문서 작성 형태는 시스템 개발에 직접적으로 관련된 문서와 개발에 간접적으로 지원되는 문서로 나눈다. <표 6>은 시스템 개발에 직접 영향을 미치는 기술문서 중 핵심이 되는 문서의 작성 현황에 대한 일부분으로 새로운 프로젝트가 진행되면서 유사한 개발환경에서는 설계문서나 세부 기능 구현에 필요한 문서들의 양이 많아지는 경향을 보였다. 이 이유는 초기 시스템설계 시 고려치 못했던 내용에 대한 기술과 추가 보완 작업이 이루어진 결과라 할 수 있다.

새로운 설계개념 도입과 개발환경의 변화를 시도한 ACE2000 프로젝트인 경우는 앞서 경험한 프로젝트에 비해 전반적으로 문서 작성량이 줄어드는 경향이 있는데 시스템의 상세설계 문서의 작성 분량은 늘어난 반면 기타 문서들은 정보의 분산화와 재활용(re-use) 및 변경(change) 기법을 사용하여 필요 시 해당 부분만 가져다가 사용하는 방법을 택했다. 이러한 예는 입·출력 메시지에 대한 데이터를 참조하면 쉽게 알 수 있다.

핵심 기술문서에 대한 작성 현황 및 소프트웨어 종합 결과는 <표 6>, <표 7>과 같다.

<표 7>의 결과를 살펴볼 때 시스템 규모(1랙 기준 2.5G→40G 증대)가 확장되는 경우에도 펌웨어 종류가 축소되어 사용되고 있는 컴포넌트 수가 별로 증가하지 않은 이유는 초기 시스템(ACE64) 개발 시 다양한 개발환경에서 개발되던 device control 용 소프트웨어인 펌웨어의 개발환경을 점차 동일한 그룹으로 통합, 관리함으로써 소스관리의 용이함과 동일 환경 하에서 응용프로그램(application program)만 교체하고 핵심(core) 부분인 operating system(os)은 통합시켜 시스템 개발기간을 단축시켰으며, 시스템의 유지보수성을 향상시켰다.

<표 8>은 각 프로젝트 별 소스라인 규모 및 에러, 품질 및 투입된 소프트웨어 인력, 개발비용 등에 대한 측정결과로 초기시스템에 비해 최종 시스템의 품질과 KLOC 당 에러 발생 수, 개발비용의 효율성, 생산성 등이 향상되고 있음을 단적으로 보여준다.

즉, 소프트웨어 개발인력에 대한 변동률은 거의 변화가 없는 반면에 프로젝트 수행 효율성이 경험의 축적 및 개발 툴의 향상, 개발환경의 개선에 따라 이전에 수행한 프로젝트에 비해 효과가 상당히 향상되고 있음을 말해준다.

<표 9> 및 <표 12>의 두 모델에 대한 프로젝트별 평가 결과는 다소 차이가 있으나 우리가 수행한 프로젝트의 성격에 적합한 COCOMO-II PA 모델과

<표 6> 기술문서 작성 현황

Project	Rel.설명서	기능규격서	기능설계서	블록규격서	블록설계서	imd	omd
ACE64	141(1.5)	324(5.79)	324(7.86)	122(6.14)	122(13.11)	376(7.5)	480(8.3)
ACE256	166(3.5)	184(9.38)	184(17.17)	144(9.31)	144(17.15)	418(3.4)	539(6.7)
ACE2000	145(2.5)	148(6.7)	148(7.83)	157(9.42)	157(17.92)	678(2)	770(2)

주) 괄호 안의 숫자는 페이지 수(평균값 적용)

<표 7> 프로젝트 별 소프트웨어 종합 결과

	ACE64	ACE256	ACE2000	Remark
Function	116종	250종	148종	S/W
Block	129개	132개	137개	S/W(TMN 제외)
Firmware	24종	19종	7종	Device 제어용
Component	7,220(개수)	7,034(개수)	8,769(개수)	S/W
SLOC	1,990,312	2,168,760	2,480,888	DB/OS/FPGA 제외

<표 8> Analysis of Software Metrics for Systems

Project	Size(KLOC)	Error	Quality	Manpower(인력)	Cost(억 원)
ACE64	977	282	0.289/KLOC	29 PY(순수S/W개발인력)	1992-98Y(6.5년) 1,178억(117,864)
ACE256	1134	194	0.171/KLOC	26 PY(순수S/W개발인력)	1998-00Y(2.5년) 246억(24,629)
ACE2000	1310	190	0.145/KLOC	22 PY(순수S/W개발인력)	2000-01(1.5년) 129억(12,948)

주 1) Cost 중 ACE64의 1998년은 50%, ACE256의 1998/2000년은 50%, ACE2000의 2000년은 50%임
2) Size는 OS/DBMS/TMN/F/W 기능은 제외(순수 응용프로그램만 적용)
3) 인력은 순수 S/W 개발 인력(H/W는 제외)
4) 총 투입비용은 10년간 1480억[148,211(백만 원)]

<표 9> 비용평가 모델 COCOMO 모델 적용 분석 결과 (단위: PM)

Project	Effort(E)			Duration(D)			Rpmp (E=MM)
	BM	IM	PA	BM	IM	PA	
ACE64	402.67	313.19	162.17	47.80	37.18	19.25	25.5
ACE256	521.3	405.46	214.83	69.02	53.69	28.45	26.0
ACE2000	711.7	553.55	299.71	111.38	86.63	46.90	22.0

주) BM: Basic Model, IM: Intermediate Model(EAF 고려, EAF=1.15 적용)
PA: Post-Architecture model, Rpmp: Real project man-power(평균값 적용)

Walston & Felix 모델을 상호 비교해보면 개발기간(D)은 COCOMO 모델이 1.15~2.5배 적게, 자원투입비(E)는 반대로 1.7~1.85배 정도 높은 것으로 추정되었다.

실제 프로젝트 수행기간(<표 12> 참조)과 비교할 경우는 개발기간은 COCOMO 모델이 자원투입은 Walston & Felix 모델이 더 근접한 것으로 추정된다. 그러나 초기시스템에 대해서는 개발기간은 반대로 된다. 정확한 데이터 추정을 위해서는 WF 모델 및 COCOMO 모델 파라미터에 대한 연구가 좀더 필요하다.

2. 소프트웨어 생산성 분석 결과

과거 초기에 개발된 소프트웨어를 가지고 소프트웨어 생산성 분석에 대한 시도(과기부 모델 및 CO-COMO-II PA 모델과의 비교, 분석)가 있었으나 지속적인 연구가 없는 실정이다[9].

소프트웨어 생산성 결과는 <표 9>에서 추정된 내용을 가지고 (22)에 적용하면 쉽게 계산할 수 있다. 추정된 결과와 실제 프로젝트에서 얻은 결과를 비교해 볼 때 COCOMO 모델보다는 Esterling 모델(EST)이 더 근접한 결과로 나타났다. 이는 대형 프로젝트에 적합한 모델로서 COCOMO 모델의 BM, IM, PA 모델이 EST보다 근접하지 못한 원인은 가격요인 및 규모요인에 대한 정확한 값의 결정에 어려움이 있기 때문인 것으로 추정된다[9]. 그밖에 정보통신부 모델에 적용하여 프로젝트 별 생산성을 분석해보면 125~130SLOC/MM 정도로 분석되었다.

- 정보통신부 모델에 의한 소프트웨어 생산성

- 기초 소요공수(MM) = (SLOC/10만)×10만 라인 기준 기초소요공수(MM)
- 보정된 소요공수(MM) = 기초소요공수 × 개발언어별 보정계수 × 규모별 보정계수 × 프로젝트 형태별 보정계수 × 적용대상 기종별 보정계수

* 소프트웨어 생산성 = 총 개발 소스(SLOC)/보정된 소요공수(MM)

P = SLOC/E, p : productivity --------(22)

FP에 의한 프로그램 생산성 평가는 4GL 기준으로 볼 때<표 10>, <표 11>에 언급된 것과 같이 추정되었다[12].

<표 11>의 내용은 ACE256/2000 프로젝트에 대한 데이터로 대형 교환시스템의 대표기능인 호 제어, 운용, 보전기능에 대한 샘플링 분석 데이터이다. 이 결과는 실제 프로젝트 수행 결과로 생성된 코드와 비교 시 프로젝트 별로 0.58~3.73배 정도 차이가 난다. 즉 기능별로 적용된 FP 수가 평균값을 적용하였기 때문에 실제 상황과는 다소 차이가 있음을 밝혀둔다.

시스템 개발 초기의 데이터를 가지고 평가한 과거 데이터와 비교 시 COCOMO-II의 PA 모델은 프로그램 생산성이 비슷한 반면 EST 모델과는 많은 차이를 보인다. 특히 자원투입량을 가지고 평가한 생산량은 상대적으로 낮아지는 경향을 보이는데 이는 현실과 맞지 않고 점차 증가되는 추세를 보이는 EST 모델이 실제 프로젝트 수행 결과와 유사한 경향을 보인 반면 경험 모델의 대표적인 LOC 모델과 비교할 때 약 2배 정도 차이가 난다. 이러한 차이는 <표 12>의 Putnam 모델에서도 발견되는데, L값(프로젝트 총 수행기간 동안 생산될 프로그램 규모)을 보면 실제 수행된 결과(real code)와 약간의 차이가 난다. 이 원인은 유사 시스템 개발에 따른 소스코드의 재활용 및 개조(ACE2000 시스템인 경우 기 완료된 ACE256 소스코드를 활용하여 소프트웨어 전

<표 10> 시스템 및 모델 별 S/W 생산성 비교 (단위: SLOC/month)

Project	BM	IM	PA	Rpmp	EST	정통부 모델
ACE64	83.6652	107.569	207.612	491.20	431.92	125.298
ACE256	83.6653	107.569	203.022	1684.62	1453.84	126.317
ACE2000	83.6649	107.569	198.676	3308.08	3308.08	126.316

주 1) PA(A=2.45, B=1.15, cost driver=0.7 적용) 모델은 COCOMO-II 모델을 의미함
2) EST: Esterling model(LOC & FP로 평가)

<표 11> Function Point 측정 모델에 의한 S/W 생산성 비교 분석 결과

버전 별	규모(SLOC)	파일 수	FP 추정치	Function category	Language	Avg. FP
SV5.1.2	14,819	24	5,748	CP (Call Processing)	CHILL	9.58
SV7.1.2	7,150	22	6,325		C/C++	11.5
SV7.1.4	11,559	29	19,495		C/C++	26.86
SV5.1.2	10,503	27	16,827	Ad (Administration)	CHILL	24.93
SV7.1.2	13,410	86	20,210		C/C++	9.40
SV7.1.4	13,918	93	22,506		C/C++	9.68
SV7.1.2	8,032	34	2,150	DH (Data Handling)	C	2.53
SV7.1.4	9,698	40	2,650		C	2.65
SV7.1.2	5,767	51	3,123	OM (Maintenance)	C/C++	2.45
SV7.1.4	7,727	59	2,802		C/C++	1.90

주 1) Avg. FP는 기능 블록 내 평균 제어문 수로 평균값임
2) FP 추정치 계산식=파일 수 × Avg. FP 수 × 25(4GL), SLOC는 실제 개발된 소스코드
3) SV5.1.2는 ACE256, sv7.1.x는 ACE2000 프로젝트에서 수행된 결과임

체에 대해 약 17.2% 정도 개조, 16.3% 재활용함), 개발경험의 축적 등에 의한 개발기간의 단축에 따른 원인으로 추정된다. 특히, ACE256인 경우는 개발 환경의 변화 없이 사용된 언어도 동일한 CHILL-Language를 사용함으로 인해 ACE64에서 개발된 코드를 거의 재사용하였으며, ACE2000 시스템은 위에 언급한 것과 같이 일부 코드(주로 호 제어 관련 프로그램)를 재활용하고 새로운 개발방법론(즉, 설계방법으로 UML(Unified Modeling Language)을 사용하고 미들웨어를 사용)을 채택하여 소프트웨어 개발 플랫폼을 통일하였다. 즉 다양한 OS 환경을 일관된 환경으로 동작시키기 위해 미들웨어를 사용하고 M&A 프로그램을 교환시스템에서 워크스테이션으로 분리, 독립시켜 개발일정을 앞당기는 방법을 채택하였다[10]-[12].

프로젝트 수행기간인 개발 일정은 실제 수행 결과가 추정된 결과보다 상당히 적게 나타나는데 ACE64인 경우에만 추정치보다 30개월 정도 더 소요된 것으로 분석되었다(<표 13> 참조).

이것은 초기 시스템 개발에 대한 경험 부족과 PSTN(Public Switched Telephone Network) 위주의 개발경험을 바탕으로 새로운 환경으로의 방향전환, 하드웨어 의존이 심한 ATM 교환 기술에 대한 기술축적미비로 인한 시행착오 반복, 시제품 개발에 따른 시범서비스 운용 후 상용제품 개발 등으로 이루어진 개발일정으로 타 프로젝트에 비해 개발기간이 상대적으로 길어졌기 때문이다. 즉, 개발기간은 단축되었으나 release 이후 사후 A/S 등을 고려할 때 실제 기간은 모델 추정치와 근사한 것으로 나타났다. 실례로 ACE256인 경우 소프트웨어 배포 후 추가 보완기간(18개월)을 가져 실제 수행기간은 약 48개월 정도이다.

Effort 적용 시 개발기간은 더욱 줄어드는 경향을 보이며 기간의 오차는 1.2배 정도 차이, 실제 수행결과와는 1.2~2배 정도의 차이를 보인다.

또 각 프로젝트 별 documentation 비율을 살펴보면 <표 6>의 핵심 기술문서에 대한 문서화 작성 현황 데이터를 참고로 하여 W&F 모델에 의해 추정하면 <표 14>와 같다. 핵심 기술문서는 시스템구조도, 요구사항 문서, 시험절차서 및 결과서, 변경 요구관리문서 등 많은 종류가 있으나 이는 전체 문

<표 12> Putnam 모델 적용 결과

Project	C_k (=good)	L(SLOC)	Real code	K	Remark
ACE64	8,000	974,140	977 K	1020.38	1st develop.
ACE256	8,000	1,129,430	1134 K	72913.7	ACE64 소스 reuse
ACE2000	8,000	1,303,870	1310 K	867319	ACE256 소스 reuse

주 1) C_k: state of technology constant
2) K: development effort for the entire life cycle

<표 13> W&F 모델 적용 결과

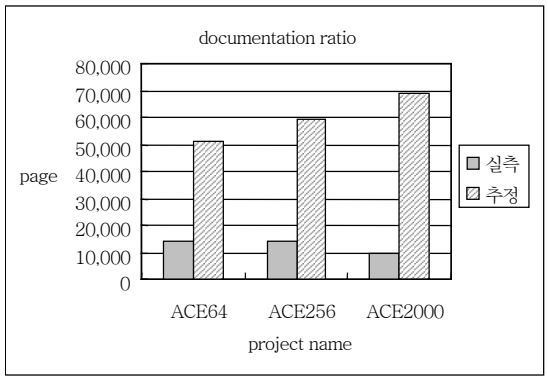
Project	E(effort)-PM	D(duration/month)	D'(effort 적용시)	S(staff)
ACE64	94.27	48.8817(78)	39.4076	0.868 (1명)
ACE256	120.43	51.5756(30)	41.323	0.8752 (1명)
ACE2000	162.29	54.3252(18)	43.2662	0.88218 (1명)

주 1) D의 괄호 안의 숫자는 실제 프로젝트 수행기간임, 스태프는 프로젝트 관리책임자를 의미함
2) PM: Person-Month

<표 14> W&F 모델에 근거한 문서 작성 비율

project	문서작성분량 (PD)	D(추정치) - page	비율(PD/D)
ACE64(1)	13,786	51,284	27%
ACE256(2)	14,308	59,615	24%
ACE2000(3)	9,701	68,966	14%

주) PD: Practice Documentation, D: Documentation



서의 30% 정도이며, 그밖에 많은 문서들 다시 말해서 직·간접으로 시스템 개발에 필요한 문서를 작성하게 된다. 즉, 프로젝트가 거듭될수록 문서화 비율은 적어지는데 이는 경험데이터 축적 및 개발기간 단축 등을 골자로 핵심 세부 설계문서 작성 분량은 늘어나는 반면에 시스템 개발에 필요한 간접적인 문서의 작성 비율을 줄여나가 시스템 개발의 효율성을 높이고 있음을 보여준다.

또한 과거에 수행한 프로젝트들이 중복적인 요소에 대해서 문서작성 비율이 높은 반면 가장 최근에 수행되고 있는 프로젝트(ACE2000)는 정보의 분산 공유 및 재활용으로 비슷한 내용의 문서 분량을 과감히 줄여 프로젝트 수행의 효율성 및 개발자의 부담을 경감시켰으며, 문서 표준화와 관리 툴의 효율적인 사용으로 과거 개발문서 작성에 많은 시간을 투자하고도 크게 활용되지 못하였던 것을 관리의 일원화를 통해 개발기간 단축 및 불필요한 사항을 줄임으로써 문서작성의 효율성을 향상시켰다. 이러한 현상은 "ATM 교환시스템"이라는 유사한 프로젝트를 연속 수행한 결과로 비슷한 기능의 구현이 많았던 점이다.

V. 결 론

다양한 소프트웨어 프로젝트 평가 모델 중 소프트웨어의 개발과정을 투명하고 효율성있게 밝힐 수 있는 모델이 생산성 평가 모델이다. 이 모델은 소프트웨어가 시간에 따라 안정화되어 가는 과정을 평가해 보는 신뢰도 성장 모델과는 달리 프로젝트 전체에 대해 그 효율성을 평가해 보는 모델로서 그 의의가 크다. 특히, 신뢰도 성장모델이 개발된 또는 개발과정에 있는 소프트웨어의 안정도 및 품질을 평가해 보는 반면 프로젝트 생산성 평가 모델은 프로젝트 전체에 대한 비용, 품질, 생산성, 문서화 상태 등을 다각도로 여러 측면을 평가할 수 있는 모델로서 실제 프로젝트를 수행하는데 실무측면에서 매우 활용이 큰 모델이라 할 수 있다. 그러나 이 모델의 단점은 여러 변수(cost driver & size factor etc.)에 대한 값의 미묘한 차이에도 그 파급효과가 크다는 것이다. 이와 같은 단점을 극복하려면 실제 프로젝트의 성격을 정확히 파악하고 모델 적용변수들에 대한 설정 값에 유의해야 한다. 즉, 과거의 유사 프로젝트 수행 시 경험한 경험 데이터들을 활용하여 정확한 추정이 되도록 노력해야 한다.

향후 연구방향은 다양한 프로젝트의 수행 결과를 정리하고 프로젝트 성격에 맞는 적합한 모델을 개발하여 차기 유사 프로젝트 수행 시 프로젝트에 대한 예측, 평가의 기초 모델로 발전시켜야 한다.

참 고 문 헌

[1] Roger S. Pressman, "Software Engineering : A Practitioner's Approach," McGRAW-HILL series in Software Engineering and technology 2nd edition, 1987, pp. 88 - 123.

[2] A.J. Albrecht, "Measuring Application Development Productivity," *Proc. IBM Applic. Dev. Symposium*, Monterey, CA, Oct. 1979, pp. 83 - 92.

[3] A.J. Albrecht and J.E. Gaffney, "Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation," *IEEE Trans. Software Engineering*, Nov. 1983, pp. 639 - 648.

[4] R. Esterling, "Software Manpower Costs: A Model," Datamation, Mar. 1980, pp. 164 - 170.

[5] C. Walston and C. Felix, "A Method for Programming Measurement and Estimation," *IBM System Journal*, Vol. 16, No. 1, 1977, pp. 54 - 73.

[6] L. Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimation Problem," *IEEE Trans. Software Engineering*, Vol. 4, No. 4, 1978, pp. 345 - 361.

[7] B. Boehm, "COCOMO-II Model Definition Manual," Univ. of Southern California, 1998.

[8] B. Boehm, Software Engineering Economics, Prentice Hall, 1981.

[9] 이재기외 2, "소프트웨어 이식성 분석과 생산성 평가," ETRI, 전자통신동향분석 제14권 제5호, 1999. 10., pp. 16 - 27.

[10] 이재기외 3, "교환소프트웨어 복잡도 연구," ETRI, 전자통신동향분석 제17권 제2호, 2002. 4., pp. 1 - 10.

[11] 이재기외 3, “미들웨어와 UML을 활용한 교환소프트웨어의 개발과 관리,” ETRI, 전자통신동향분석 제16권 제5호, 2001. 10., pp. 49 – 60.

[12] 이재기외 3, “통신 소프트웨어의 복잡도 분석 사례 연구,” 대한전자공학회 하계학술대회논문집 1권, Vol. 25, No. 1, 2002. 6., pp. 409 – 412.