가중치를 적용한 FFP 소프트웨어 규모 측정

A Software Size Estimation Using Weighted FFP

박 주 석*
Juseok, Park

요 약

대부분 소프트웨어 규모 추정 기법들은 사용자에게 제공될 기능에 기반을 두고 있으며, 기능에 대한 점수를 부여하는 과정에서 복잡도를 함께 고려하고 있다. 완전기능점수 기법은 데이터 처리, 실시간 시스템과 알고리즘 소프트웨어 등 광범위한 분야에 적용되는 장점을 갖고 있는 반면에 규모를 추정하는데 필요한 기능 요소들에 대한 가중치를 부여하지 않는 단점도 갖고 있다. 본 논문은 신규로 개발되는 프로젝트와 유지보수 프로젝트들에 적용되는 완전기능점수 계산 방법에 각 기능요소들에 대한 복잡도를 고려하여 소프트웨어 규모를 추정할 수 있는 방법을 제안하였다. 이를 위해 기능 점수 기반으로 실측된 데이터를 이용하여 제안된 방법의 타당성을 검증하였다. 검증한 결과, 소프트웨어의 규모 추정에 사용되는 속성들인 기능 요소들에 다른 가중치를 적용하였을 경우 보다 좋은 규모 추정이 가능하였다.

Abstract

Most of the methods of estimating the size of software are based on the functions provided to costumers and in the process of granting the score to each function we consider the complexity during the process. The FFP technique has advantages applied to vast areas like data management, real-time system, algorithmic software, etc, but on the other hand, has disadvantage on estimating sizes for weights for necessary function elements. This paper proposes the estimating method for software size by considering the complexity of each function elements in full function point calculation method applied to a new developed project and maintenance projects. For this, based on function point by using surveyed data proved the validity of proposed method. The valid result, was that the function elements, the attributes used in size estimation of software, estimated better estimated sizes than in the case of other weights being applied.

☞ Keyword : Software Size, Function Point, Full Function Point, FFP, Complexity-Weighted, Functional Elements

1. 서 론

소프트웨어를 개발시 소요되는 노력, 기간과 비용을 예측하기 위해서는 소프트웨어의 크기를 표현할 수 있는 근본적인 단위가 필요하다. 소프트웨어 측정분야에서는 규모를 소프트웨어의 근본적인 단위로 채택하였다.[1] 소프트웨어 규모(Size)를 표현할 수 있는 속성으로 Mišić[1], Mcphee[2]는 복잡도, 기능, 길이와 재사용으로, Sultanoglu[3], Penton과 Pfleeger[4]는 복잡도, 기능성과 길이로 정의하였다. 길이에 기반한 소프트웨어 규모 추정 방법은 라인 수로 대표적으로 COCOMO 모델[5]이 있다. 기능에 기반한 규모 추정방법으로는 기능점수(Function Point, FP), 완전 기능점수(Full Function Point, FFP), COCOMO II, 쓰임새 점수(Use Case Point, UCP), 특징점수(Feature Point)와 객체점수(Object Point) 등이 있다.[6-10] 또한, 복잡도 추정방법에는 Halstead의 Software Science와 McCabe의 Cyclomatic Complexity가 있다. 이들 방법 중 실제로 라인 수와 기능점수 방법이 가장 많이 적용되고 있다.[11] 라인 수는 사후 추정 기법이고 기능점수는 사전 추정 기법이다. 개발 초기에 소프트웨어의 개발계획 작성과

* 정 회 원 : 국방대학교 직무연수부 교수
iage2k@dreamwiz.com(제 1저자)
[2004/09/06 투고 - 2004/10/01 심사 - 2004/12/01 심사 완료]

계약에 활용하기 위해 기능점수 기법을 보다 많이 적용하고 있다.[12] 기능점수 방법은 사용자 관점에서 기능으로 소프트웨어 규모를 정량화하는 방법으로 경영정보시스템 소프트웨어에 기반을 두고 개발되어 실시간이나 공학계산 소프트웨어에는 적용이 불가능하게 되었다. 1998년 COSMIC(Common Software Management International Consortium)이 설립되어 MIS의 데이터 관리 위주인 기능점수 기법을 제어관리 위주인 실시간 시스템과 내장형 소프트웨어로 적용범위를 확장하였다. 이를 완전기능점수 또는 COSMIC-FFP라 부른다. COSMIC-FFP는 소프트웨어 기능 규모 측정에 대한 ISO-14143 표준을 만족하도록 설계되어 국제 표준화인 ISO-19761[8]로 등재되었으며, ISBSG(International Software Benchmarking Standard Group)에 의해서도 인정되었다[13].

기능 속성에 기반한 대부분의 소프트웨어 규모 측정방법들은 기능을 측정하는 과정에서 복잡도를 함께 고려하고 있다. 소프트웨어의 복잡도는 시스템 복잡도와 컴포넌트 복잡도로 구분될 수 있다. 그러나 COSMIC-FFP 기법은 소프트웨어 기능 프로세스의 컴포넌트 관점을 기술하고 있으나 규모 추정과 관련된 기능 요소들에 가중치를 부여하지 않고 있다.[14]

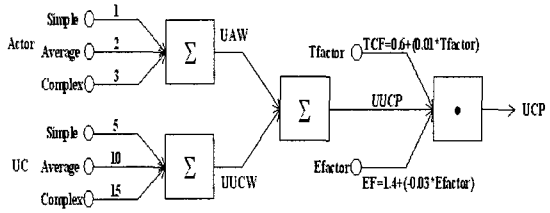
본 논문은 COSMIC-FFP가 내포하고 있는 문제점을 해결하여 보다 현실적으로 타당하게 적용할 수 있는 복잡도 가중치를 고려한 FFP 기법을 제안하고 적합성을 평가해 본다. 2장에서는 기존의 소프트웨어 규모 추정 방법들을 살펴본 다음, 3장에서는 개선된 완전기능점수 모델을 제안하고, 4장에서 제안된 모델의 적합성을 평가해 본다.

2. 기존의 소프트웨어 규모 추정 방법

소프트웨어 규모는 개발비용과 생산성을 평가하는 주요 인자로 사용된다.[12] 대부분의 소프트웨어 규모 측정 방법들은 소프트웨어의 규모와 복잡도를 고려하고 있다. 이들 부류가 쓰임새 점수, 기능 점수, COCOMO II 등이다. 이들 부류와는 다르게 복잡도를 전혀 고려하지 않는 방법으로 COSMIC-FFP가 있다. 본 장에서는 이들 UCP, FP와 COSMIC-FFP 방법을 간략히 살펴본다.

2.1 쓰임새 점수

쓰임새 점수는 기능점수 기법을 기반으로 제안되었으며, 그림 1과 같이 계산된다.[10]



〈그림 1〉 UCP 계산과정

첫 번째로, 액터 복잡도(Simple, Average, Complex)에 가중치를 부여하여 UAW(Unadjusted Actor Weights)를 계산한다. 두 번째로, Transaction이나 Class의 수에 따라 복잡도(Simple, Average, Complex)가 결정되고 이에 따라 가중치가 부여되어 UUCW(Unadjusted Use Case Weights)가 계산된다. 세 번째로, 식(1)의 UUCP(Unadjusted Use Case Points)를 계산한다.

$$UUCP = UAW + UUCW \quad (1)$$

네 번째로, 표 1의 TFactor 측정하여 식(2)에 의해 생산성에 영향을 미치는 기술 복잡도 요인(Technical Complexity Factor, TCF)을 계산한다.

$$TCF = 0.6 + (0.01 \cdot TFactor) \quad (2)$$

다섯 번째로, 표 2의 Efactor 측정으로 식(3)에 의해 생산성에 영향을 미치는 환경 복잡도 요인(Environmental Complexity Factor, EF)를 계산한다.

〈표 1〉 TFactor 계산

기술 요인	Description	Weight Factor (①)	Assigned Value (②)	TF_i (①x②)
T1	분산시스템	2	0 ~ 5	
T2	응답시간 또는 처리성능	1		
T3	사용자 능력	1		
T4	내부처리 복잡도	1		
T5	코드 재사용성	1		
T6	설치 용이성	0.5		
T7	사용 용이성	0.5		
T8	휴대용	2		
T9	변경 용이성	1		
T10	병렬처리	1		
T11	특수 보안	1		
T12	제3자 직접 접근성	1		
T13	특별한 사용자 교육시설	1		
TFactor (Total Technical Factor)= $\sum_{i=1}^{13} TF_i$				

$$EF = 1.4 + (-0.03 \cdot EFactor) \quad (3)$$

여섯 번째로, 식(4)의 조절된 쓰임새 점수(Adjusted Use Case Point, AUCP)를 계산한다. AUCP를 UCP라 칭한다. TFactor와 EFactor에서 적용되는 가중치 할당 값은 기능점수의 DI 계산에 적용된 기준과 동일하다.

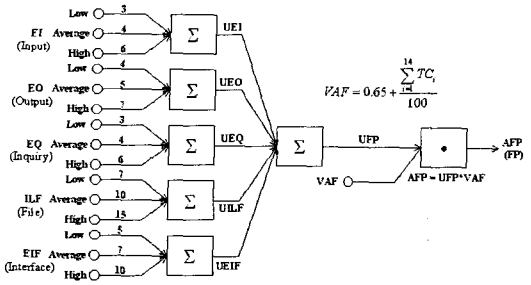
$$AUCP = UUCP \times TCF \times EF \quad (4)$$

2.2 기능점수

사용자에게 제공되는 응용프로그램의 규모를 계산 가능한 기능으로 측정하기 위해 입력(External Inputs, EI), 출력(External Outputs, EO), 조회(External Inquiries, EQ), 화일(Internal Logical Files, ILF)과 인터페이스(External Interface File, EIF) 의 기능 요소로 세분화하여 그림 2와 같이 계산한다.[6]

〈표 2〉 EFactor 계산

환경 요인	Description	Weight Factor (①)	Assigned Value (②)	EF_i (①x②)
E1	RUP 친숙도	1.5	0 ~ 5	
E2	해당분야 경험	0.5		
E3	객체지향 경험	1		
E4	분석책임자 능력	0.5		
E5	동기부여	1		
E6	요구사항 안정 정도	2		
E7	비상근 개발요원	-1		
E8	프로그램 언어 어려움	-1		
EFactor (Total Environmental Factor)= $\sum_{i=1}^{8} EF_i$				



〈그림 2〉 기능점수 계산과정

첫 번째로, ILF와 EIF는 데이터 요소(Data Element Type, DET) 수와 레코드 요소(Record Element Type, RET) 수에 따라 복잡도(Low, Average, High)가 결정되며, EI, EO와 EQ는 DET와 참조하는 파일(RFT) 수에 따라 복잡도를 계산한다. 두 번째로, 각 기능요소(EI, EO, EQ, ILF, EIF)에 대해 복잡도별로 가중치를 부여하여 표 3에 따라 조절이 안된 기능점수(UFP)를 계산한다.

세 번째로, 사용자에게 제공되는 응용프로그램의 일반적인 시스템 속성을 <표 4>에 의해 평가한 후, 식(5)의 VAF(Value Adjustment Factor)를 계산한다.

$$VAF = (TDI \cdot 0.01) + 0.65 \qquad (5)$$

네 번째로, 식(6)에 의해 소프트웨어 프로젝트에 대한 최종 조절된 기능점수(AFP)를 계산한다. AFP를 일반적으로 기능점수(FP)라 칭한다.

$$AFP = VAF \cdot UFP \qquad (6)$$

2.3 완전기능점수

COSMIC-FFP는 그림 3과 같이 기능 요구사항에 기반을 둔 일반적인 소프트웨어 모델이다. 이 모델에 의하면 소프트웨어의 생명주기 초기 단계에서도 소프트웨어는 사용자의 기능 요구사항들로 취급하고 있으며, 이들은 기능 프로세스로 구현된다. 기능 프로세스는 다시 데이터 이동형과 데이터 처리형으로 구성된다. 기능 규모를 측정하기 위해 그림 4와 같이 기능 프로세스는 하나의 자극을 주는 사건에 의해 활성화되기 때문에 데이터 처리형은 취급하지 않고 데이터 이동형만을 고려하였다. 자극이 주어지면 입력 데이터(Entry, Read)를 받아들여 데이터를 처리하여 출력 데이터(Write, Exit)를 생성한다.

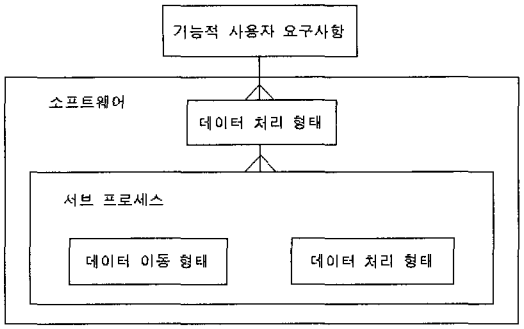
COSMIC-FFP는 데이터 이동형을 4개의 속성으로 분류한다. 소프트웨어 측면에서 바라볼 때, 입구와 출구는 사용자와 데이터 속성을 주고받는

〈표 3〉 UFP 계산

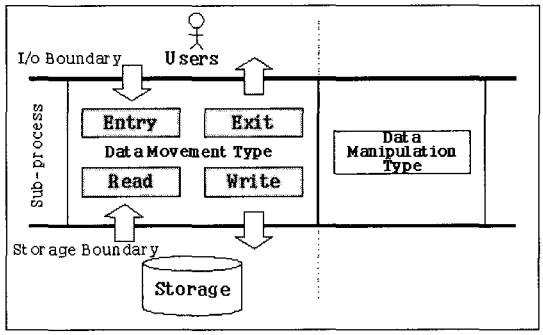
기능 요소	기능 복잡도			Total (①+②+③)
	Low (①)	Average (②)	High (③)	
EI	__ × 3 = _	__ × 4 = _	__ × 6 = _	
EO	__ × 4 = _	__ × 5 = _	__ × 7 = _	
EQ	__ × 3 = _	__ × 4 = _	__ × 6 = _	
ILF	__ × 7 = _	__ × 10 = _	__ × 15 = _	
EIF	__ × 5 = _	__ × 7 = _	__ × 10 = _	
UFP (Unadjusted Function Points)				

〈표 4〉 TDI 계산

GSC	DI(Degree of Influence)	Remark
데이터 통신		[DI] 0 : 영향 없음 1 : 우발적 영향 2 : 보통의 영향 3 : 평균적인 영향 4 : 중요한 영향 5 : 완전히 강한 영향
분산 데이터 처리		
성능		
가중된 사용 형상		
처리율		
온라인 데이터 입력		
사용자 능률		
온라인 갱신		
복잡한 처리		
재사용성		
설치 용이성		
운영 용이성		
다중 설치운영		
변경 용이성		
TDI(Total Degree of Influence)= $\sum_{i=1}^{14} DI_i$		



〈그림 3〉 COSMIC-FFP의 일반적인 소프트웨어 모델[1]



〈그림 4〉 COSMIC-FFP의 서브 프로세스 형[1]

것이며, 읽기와 쓰기는 저장장치와 데이터 속성을 주고받는 형태이다. COSMIC-FFP에 기반한 소프트웨어 규모의 단위는 Cfsu(Cosmic Functional Size Unit)로 표기한다. 신규로 개발되는 경우, 주어진 기능 프로세스의 기능 규모는 식(7)을 사용하여 Cfsu로 계산된다.

$$\text{FP Size} = \sum(\text{Ne}*\text{Eus}) + \sum(\text{Nx}*\text{Xus}) + \sum(\text{Nr}*\text{Rus}) + \sum(\text{Nw}*\text{Wus}) \quad (7)$$

여기서, Ne는 입구의 수, Eus는 입구 단위 크기(Unit Size), Nx는 출구의 수, Xus는 출구 단위 크기, Nr은 읽기의 수, Rus는 읽기 단위 크기, Nw는 쓰기의 수, Wus는 쓰기 단위 크기를 의미한다.

기존에 개발된 기능 프로세스의 변경이 발생한 경우는 식(8)에 의해 기능규모가 계산된다.

$$\text{Size}_{\text{Cfsu}}(\text{Change}) = \sum \text{size}(\text{Added}) + \sum size(Changed) + \sum \text{size}(\text{Deleted}) \quad (8)$$

개발이 완료된 소프트웨어를 운영하면서 발생하는 유지보수 활동은 정정 유지보수(Corrective Maintenance), 완전 유지 보수(Perfective Maintenance)와 적응 유지 보수(Adaptive Maintenance)로 구분된다. 정정 유지보수는 운영 단계에서 발견된 결함을 제거하는 활동이며, 새로운 기능을 추가하여 성능을 향상시키는 경우를 완전 유지 보수라 한다. 이에 비해, 데이터베이스 또는 운영체제의 업그레이드, 컴파일러 버전 변경 등과 같은 운영환경 변화로 요구되는 소프트웨어 수정을 적응 유지 보수라 한다. 운영단계에서 유지보수로 인해 기존 프로그램의 기능들이 추가(Added), 변경(Changed) 또는 삭제(Deleted) 되는가에 관심을 가진다. 따라서 식(8)은 유지보수 프로젝트라 할 수 있으며, 이에 비해 식(7)은 순수한 개발프로젝트로 생각할 수 있다.

복잡도를 연구하는 분야에서는 소프트웨어 복잡도가 소프트웨어의 개발비용을 추정하기 위한 중요한 척도라고 가정한다. Sellers[15]는 비용추정에 규모보다는 복잡도가 개발비용 추정에 보다 관련이 있는 속성이라고 가정하였다. 소프트웨어의 복잡도는 기능프로세스의 내부 복잡도인 컴포넌트 복잡도(Component Complexity)와 시스템 측면에서의 복잡도(System Complexity)로 구분될 수 있다. 기능에 기반을 둔 소프트웨어 규모 추정 기법들을 살펴보자. 이들 기법들의 공통점은 첫 번째로, 컴포넌트 복잡도와 시스템 복잡도를 모두 고려하고 있다. 두 번째로, 컴포넌트들 간의 복잡도는 고려하고 있지 않다.

먼저 컴포넌트 복잡도 측면에서 살펴보면 기능점수 기법은 입력, 출력, 조회, 파일, 인터페이스들의 복잡도를, 쓰임새 점수는 액터와 클래스 또는 트랜잭션에 대한 복잡도를 고려하였다. 시스템 복잡도 측면에서, 기능점수 기법은 14가지의 일반적인 시스템 속성과 가치조절인자를 고려하였으나

쓰임새 점수는 기술적 복잡도와 환경적 복잡도를 고려하였다. 즉, 동일한 규모일지라도 속성들에 따라 개발에 소요되는 노력은 차이가 발생할 수 있음을 의미한다. 따라서, 소프트웨어 개발에 소요되는 노력, 비용 또는 개발기간 추정의 정확성을 향상시키기 위해서는 복잡도가 고려되어야만 한다.

신규 개발 프로젝트의 경우, Monaka et al.[14]은 GUI(Graphic User Interface)에 대해 입력 컴포넌트는 6.0, 출력 컴포넌트는 1.0, 입력과 출력이 복합된 컴포넌트는 3.0의 가중치를 부여하는 연구 결과를 발표하였다. 유지보수 프로젝트에 대한 기능점수 추정시 추가, 수정, 삭제에 대해 표 5와 같이 다양한 가중치를 부여하여 소프트웨어 규모를 측정하였다.

〈표 5〉 유지보수 소프트웨어 규모 대상 선정 방법

구분	모델	가중치		
		추가	수정	삭제
가중치 미 적용	COCOMO[5]	1.0	1.0	-
	IFPUG[6]	1.0	1.0	1.0
가중치 적용	NESMA[16]	1.0	0.25~1.5	0.4
	Cote et al.[17]	1.0	0.8	0.33
	권기태 et al.[18]	1.0	0.8	0.33

다음으로, 시스템 복잡도를 고려하여 보자. IFPUG 지침서[6]는 VAF 계산 단계를 기능점수 계산과정의 마지막 부분에서 언급하고 있다. 이와 같이 하는 이유로서 첫 번째는 기능점수를 계산하는 실무자는 VAF에 대한 평가의 어려움으로 인해 계산을 수행하는 시점을 계속 지연시키려고 하기 때문이다.

다음으로 시스템 전문가가 시스템의 특정한 특징만으로는 기능점수를 계산하는 것이 충분하지 않음을 제기하는 일이 종종 발생하며, 기능점수를 계산하는 실무자는 이 특징을 VAF가 계산될 때 반영할 것을 요구하고 있기 때문이다. 따라서, VAF는 전체적인 시스템의 복잡도를 파악해 주는 것으로 시스템 전문가의 도움을 받아 보다 정확한 계산을 할 수 있다.[19] 결과적으로 VAF의 값에 따라 실제 계산된 AFP는 UFP에 대해 ±35%의 편차를 가진다. 따라서, VAF를 잘못 측정하였을 경우 최종적으로 계산된 소프트웨어 규모에 심한 오차를 유발시킬 수 있다. 기능점수 방법을 적용하는 실무자들은 UFP가 AFP와 같이 정확성을 가지고 있기 때문에 VAF 값을 무시하고 있다.[7] 또한, Kitchenham과 Känsälä[20]도 VAF 값이 개발노력의 추정 능력을 향상시키지 못함을 보였다. 이에 따라 Ribu[10]도 AUCP 계산에서 TCF를 제외시켜도 개발노력을 추정하는데는 영향이 없음을 제시하였다. 따라서, 시스템 복잡도는 고려하지 않는 것이 보다 올바른 선택이 될 수 있다.

3. 복잡도 가중치가 적용된 완전기능점수

복잡도를 고려한 FFP 연구 결과는 Tran-Cao[12]와 Monaka et al.[14]가 있다. Tran-Cao[12]는 기능 프로세스들 사이의 관계인 시스템 복잡도로 제어와 데이터 통신(두 프로세스 사이의 통신에 관계된 데이터 그룹의 수), 병행(시스템에 병행적으로 접근하는 데이터 그룹의 수)와 이벤트 또는 인스턴스 취급(다중 인스턴스를 가지는 프로세스의 수)를 고려하였다. 또한, 기능 프로세스 내부의 복잡도로, 입-출력의 다른 경우(프로세스의 다른 경우의 수)와 데이터 이동(입구, 출구, 읽기와 쓰기의 수)을 고려하였다. 이에 따라 소프트웨어의 기능 복잡도는 이들 5가지 속성들의 함수로 표현될 수 있음을 제시하였다. Monaka et al.[14]는 170개의 화면으로 구성된 78,000 라인의 자동차 항법 소프트웨어에 대해 운전자와 상호작용하는 부분만을 고려하여 기능 규모 측정을 위해 COSMIC-FFP를 적용하였다. 소프트웨어를 입력 컴포넌트, 출력 컴포넌트와 혼합 컴포넌트로 구분하여 각각에서 발생하는 입구, 출구, 읽기와 쓰기에 가중치를 다르게 주었을 경우 개발노력을 보다 정확히 추정할 수 있다고 제안하였다. 그러나 이들 두

가지 제안된 이론 모두 실측 데이터의 부족으로 적절한 가중치 값의 설정은 현실적으로 어려움을 제기하고 있다. 또한, 기존의 COSMIC-FFP 기반 개발비용 추정에 관한 연구로는 Abran et al.[21]과 Levesque et al.22]이 있다. 이들 모델들을 유도하기 위해 수집된 데이터들을 살펴보면 소프트웨어 규모는 독립적으로 입구, 출구, 읽기와 쓰기들의 개수가 표현되지 않고 단지 이들 속성들을 합한 Cfsu로만 명기되어 있다. 따라서 이들 데이터들로부터 입구, 출구, 읽기와 쓰기 속성들에 대한 정확한 가중치를 찾는 것은 불가능하다.

현 시점에서는 신규 개발 프로젝트에 대해 실제 수집될 수 있는 데이터들이 없기 때문에 복잡도 가중치를 부여한 제안된 모델의 타당성을 실증적으로 검증할 수 는 없으나 이론적으로 가중치를 부여하는 것이 보다 적합함을 알 수 있다. 그러므로 유지보수 프로젝트에 대해 복잡도 가중치가 연구된 표 5에 기반하여 기존의 COSMIC-FFP의 소프트웨어 규모 추정방법에 가중치를 부여한 기법을 제안한다. 입구, 출구, 읽기와 쓰기 속성들에 대한 복잡도를 각각 Ce, Cx, Cr, Cw라 하자. 또한 추가, 변경과 제거 속성들에 대한 가중치를 각각 Ca, Cc, Cd라 하자. 먼저, 컴포넌트 복잡도를 고려하여 보면 신규로 개발되는 소프트웨어의 규모에 대해서는 식(7)이 (9)로, 유지보수 소프트웨어의 규모에 대해서는 식(8)이 (10)으로 변경된다.

$$\mathrm{Size}_{\mathrm{Cfsu}}(\mathrm{New\ Developed}) = \sum(\mathrm{Ne}*\mathrm{Eus}*\mathrm{Ce}) + \sum(\mathrm{Nx}*\mathrm{Xus}*\mathrm{Cx}) + \sum(\mathrm{Nr}*\mathrm{Rus}*\mathrm{Cr}) + \sum(\mathrm{Nw}*\mathrm{Wus}*\mathrm{Cw}) \quad (9)$$

$$\mathrm{Size}_{\mathrm{Cfsu}}(\mathrm{Enhanced}) = \sum\mathrm{Added}(\mathrm{Na}*\mathrm{Ca}) + \sum\mathrm{Changed}(\mathrm{Nc}*\mathrm{Cc}) + \sum\mathrm{Deleted}(\mathrm{Nd}*\mathrm{Cd}) \quad (10)$$

제안된 모델과 기존의 COSMIC-FFP의 차이점은 첫 번째로, COSMIC-FFP의 컴포넌트 복잡도를 고려하지 않는 문제점을 개선하였다. 두 번째로 기존 모델은 소프트웨어 규모를 추정하는데 필요한 속성들을 모두 합한 값으로 단변량인데 비해 제안된 모델은 속성들 각각이 의미를 갖도록 함으로써 다변량으로 취급할 수 있다는 점이다.

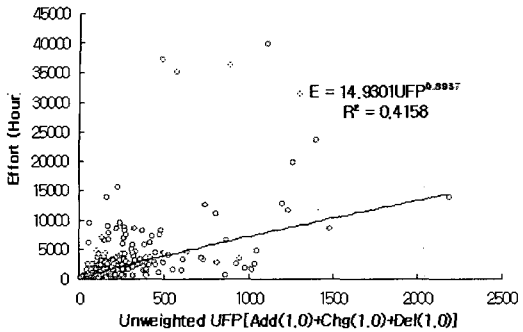
4. 제안된 모델 적용 / 평가

유지보수 프로젝트의 경우, COSMIC-FFP와 FP 모두 속성들이 어떤 것인지에 관계없이 추가, 변경과 삭제가 되었는지에만 관심을 갖고 있다. 따라서 유지보수 프로젝트에 대해 제안된 모델의 타당성을 기능점수 데이터들을 이용해 검증해 본다. 검증 방법의 초점은 제안된 모델에 대한 적절한 가중치 값들을 찾고자 하는 것이 아니라 가중치를 부여하였을 경우와 가중치를 부여하지 않았을 경우 어떤 경우가 추정된 소프트웨어 규모에 따라 개발노력을 보다 정확히 예측할 수 있는가 측면이다. 모델 평가에는 ISBSG이 보유한 Release 6[23]의 유지보수 프로젝트 234개를 대상으로 하였다. 모델을 평가하기 위해 표 6과 같이 다양한 가중치를 적용한다.

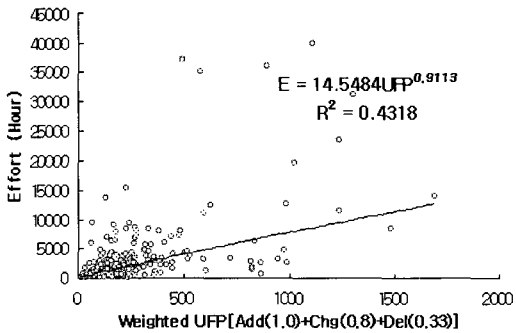
〈표 6〉 모델 평가를 위한 가중치 적용 사례

구분	관련 연구	가중치		
		추가	수정	삭제
가중치 미 적용	COCOMO[9], IFPUG[6]	1.0	1.0	1.0
가중치 적용	Cote et al.[17], 권기태 et al.[18]	1.0	0.8	0.33
	-	2.0	1.0	0.5
		3.0	2.0	1.0
		5.0	3.0	1.0

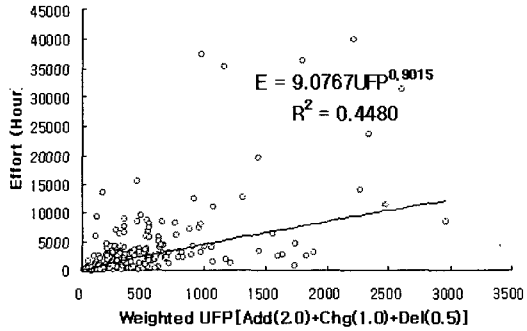
다음으로, 유지보수에 투입되는 노력을 추정하기 위해 선형, 로그, 다항식, 누승, 지수 등 다양한 회귀모델들 중에 가장 적합한 모델을 선택하였다. 회귀분석 결과 가중치가 적용되지 않은 경우와 가중치가 적용된 경우의 소프트웨어 규모에 따른 개발노력 추정 결과는 그림 5에 제시하였다.



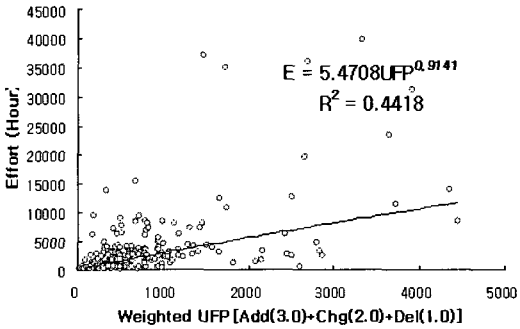
(a) 가중치를 적용하지 않은 경우



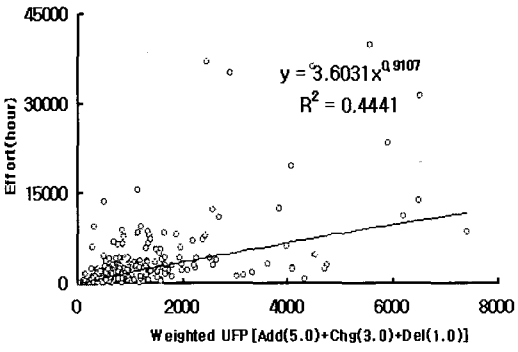
(b) 추가(1.0), 변경(0.8), 삭제(0.33) 가중치가 주어진 경우



(c) 추가(2.0), 변경(1.0), 삭제(0.5) 가중치가 주어진 경우



(d) 추가(3.0), 변경(2.0), 삭제(1.0) 가중치가 주어진 경우



(e) 추가(5.0), 변경(3.0), 삭제(1.0) 가중치가 주어진 경우

〈그림 5〉 소프트웨어 규모에 따른 개발노력 추정

모델 평가 기준으로 결정계수($0 \leq R^2 \leq 1$)와 MMRE(Mean Magnitude of Relative Error)를 적용하였다. 주어진 데이터들의 변동을 적절히 표현할 수 있는 모델을 찾기 위해 결정계수를 이용한다. 종속변수의 값(노력)은 독립변수의 값(소프트웨어 규모)에 의해 결정되는 부분과 미지의 오차의 합으로 나타나며, 총 변동(주어진 데이터의 실제 노력 값) 중에서 회귀모델에 의해 설명되는 변동 비율이 결계수이다. 결정계수 값이 클수록 쓸모 있는 회귀직선인 모델이 얻어진다.[12] 비록 결정계수에 의해 좋은 모델이 얻어졌더라도, 주어진 데이터들이 값이 큰 부분에 약간의 이상점들이 분포한 상태에서 회귀직선이 이 이상점들을 잘 표현하고 값이 작은 표본에 대해서는 표현을 하지 못할 경우에도 결정계수는 큰 값을 나타낼 수 있다. 이 경우 좋은 모델이라고 판단할 수 없다. 따라서, 모델 정확도를 평가하는 측도로 MMRE가 적용된다. RE

$= \frac{\text{추정치} - \text{실측치}}{\text{실측치}}$, MRE(Magnitude of RE) = $|RE|$, MMRE(Mean MRE) = $100/n * \sum_{i=1}^{n} MRE$. MMRE가 작은 값이면 모델은 평균적으로 좋은 모델임을 알 수 있다. 표 7과 같이 결정계수와 MMRE를 측정한 결과 가중치를 적용한 3가지 모델 모두 가중치를 적용하지 않은 경우보다 좋은 성능을 나타내고 있다.

〈표 7〉 모델 성능 비교

구분	가중치			모델 성능	
	추가	수정	삭제	결정계수	MMRE
가중치 미 적용	1.0	1.0	1.0	0.4158	100.37
가중치 적용	1.0	0.8	0.33	0.4318	99.08
	2.0	1.0	0.5	0.4480	98.18
	3.0	2.0	1.0	0.4418	98.26
	5.0	3.0	1.0	0.4441	97.35

5. 결 론

소프트웨어를 개발하는데 소요되는 노력, 비용과 기간을 개발 초기에 예측하기 위해서는 소프트웨어를 정량화하는 것이 필수적으로 규모로서 길이(라인 수), 기능 또는 복잡도와 같은 속성들로 표현된다. 대부분의 소프트웨어 규모 추정 방법들이 사용자에게 제공될 기능에 기반을 두고, 기능에 대한 점수를 부여하는 과정에서 복잡도를 함께 고려하고 있다.

특히, 완전기능점수 기법은 데이터 위주의 경영정보 시스템, 실시간 시스템뿐만 아니라 알고리즘 위주의 과학계산용 소프트웨어 등 광범위한 분야에 적용되도록 고안되었다. 이러한 장점에도 불구하고 가장 중요한 단점 중 하나는 기능을 표현하는 기능 요소들에 가중치를 부여하지 않고 있다는 것이다. 이는 내부 파일에 자료 저장과 내부 파일로부터 자료 검색, GUI 입력화면, 데이터 출력 등과 관련된 기능 프로세스들을 개발하는데 모두 동일한 노력이 소요됨을 의미하나 현실적으로 동일한 노력이 소요되지 않는다는 것이 기존의 기능점수, COCOMO II, 쓰임새 점수 등 소프트웨어 규모 추정 방법론에서 각각에 복잡도를 고려함을 알 수 있다.

본 논문은 완전기능점수 기법에 가중치를 적용하는 방안을 제안하였다 . 먼저, 신규로 개발되는 프로젝트의 경우에 대해 가중치를 부여한 완전기능점수 계산 방법을 제안하고, 다음으로 유지보수 프로젝트에 대해서도 가중치를 부여한 완전기능점수 기법을 제안하였다. 제안된 이론은 기능점수로 획득된 실제 데이터들을 이용하여 가중치를 부여한 경우가 개발에 투입되는 노력을 예측하는데 보다 향상된 결과를 나타냈다.

제안된 방법에서 실제 수행된 프로젝트들로부터 얻을 수 있는 자료부족으로 제안된 이론의 충분한 검증과 타당한 가중치의 값을 유도하지 못했다. 그러나 20여 년 동안 보완 발전된 기능점수 기법으로부터 가중치를 적절히 부여할 수 있는 방법을 모색해야 할 것이다.

참 고 문 헌

[1] V. B. Mišić, "Software Size and Cost Estimation," Department of Computer Science, University of Belgrade, 2003.

[2] C. Mcphee, "SENG 621-Software Process Management," University of Calgary, 1999.

[3] S. Sultanoglu, "Software Measurement," Department of Computer Science & Eng., Hacettepe University, 1998.

[4] N. E. Fenton and S. L. Pfleeger, "Software Metrics: A Rigoous and Practical Approach, 2nd Edition, PWS Publishing Company, 1997.

[5] B. W. Boehm, "Software Engineering Economics," Prentice-Hall, 1981.

[6] M. Bradley, "Function Point Counting Practices Manual, Release 4.1," International Function Point Users Group(IFPUG), 1999.

[7] C. Symons, "COSMIC-FFP Measurement Manual, Version 2.2 (The COSMIC Implementation Guide for ISO/IEC 19761: 2003)," Common Software Measurement International Consortium, 2003.

[8] ISO/IEC FDIS 19761, "Software Engineering-COSMIC-FFP-A Functional Size Measurement Method," 2002.

[9] B. W. Boehm et al, "Software Cost Estimation with COCOMO II," PrantIce-Hall, 2000.

[10] K. Ribu, "Estimating Object-oriented Software Projects with Use Cases," University of Oslo Department of InformatIcs, Master of Science Thesis, 2001.

[11] R. E. Park, "Software Size Measurement: A Framework for Counting Source Statements," Technical Report CMU/SEI-92-TR-020, 1992.

[12] D. Tran-Cao, A. Abran, and G. Lèvesque, "Functional Complexity Measurement," International Workshop on Software Measurement (IWSM '01), 173 Montral, Qubec, Canada, 2001. 8.

[13] HierarchyMaster, "Software Metrics," http://www.hmaster.com/FFP/metrics.html, HierarchyMaster, Pty, Ltd., Australia, 2003.

[14] N. Monaka, A. Kakural, E. Bukhary, and M. Azuma, "A Complexity-Weighted Functional Size Metric for Interactive Software," Advanced Institute for Science & Eng., Waseda University, 2002.

[15] H. Sellers, "Object-Oriented Metrics-Measures of Complexity," Prentice Hall, New Jersey, 1996.

[16] NESMA, "Function Point Analysis for Software Enhancement," NESMA, 2001.

[17] Cote and St-Pierre, "A Model for Estimating Perfective Software Maintenance Projects," Proceedings of Conference on Software Maintenance, Vol. 11, pp. 328-334, 1990.

[18] 권기태, 신수정, "소프트웨어 유지보수 비용산정 모델 개선," 한국정보과학회 소프트웨어공학회 지 제16권 제2호, pp. 31-42, 2003.

[19] "Function Point FAQ," Software Composition Technologies, Inc. 1997.

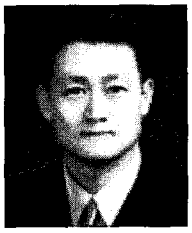
[20] B. Kitchenham and K. Känsälä, "Inter-item Correlation Among Function Points," National Computing Center Ltd, UK and VTT, Finland, 1997.

[21] A. Abran, C. Symons, and S. Oligny, "An Overview of COSMIC-FFP Field Trial Results," ESCOM 2001, London, England, 2001.

[22] G.. Levesque and V. Bevo, "Comparing COSMIC-FFP and SLIM Back-Firing Function Points Size Measurements," ICSSEA, 2001.

[23] ISBSG, "Worldwide Software Development-The Benchmark Release 6," Victoria, Australia International Software Benchmarking Standards Group, 2000.

◐ 저 자 소 개 ◑



박 주 석 (Juseok, Park)
1984년 해군사관학교 전자공학과 졸업(공학사)
1995년 국방대학원 전자계산학과 졸업(전산학 석사)
2004년 숭실대학교 일반대학원 컴퓨터학과 졸업(공학박사)
2001년~현재 국방대학교 직무연수부 교수
관심분야 : 비용산정, 표준 및 프로세스, 정보체계사업관리, 소프트웨어 품질보증, 정보전
E-Mail : iage2k@dreamwiz.com