외국기사소개

소프트웨어 규모 견적 기술의 동향†

西 山 茂††

❖ 목 차 ❖

1. 서 론

소프트웨어 개발의 초기 단계에서 각종 지식을 토대로 여러 가지 사항을 예측해서 그에 필요한 수단을 강구해 두는 것은 높은 품질을 가진 소프트웨어의 효율적인 개발에 필수적인 사항이다. 소프트웨어 개발에 있어서 예측의 대상이 되는 것으로는 크기(량 또는 규모), 투입하는 工數, 개발 기간, 개발에 사용되는 기술, 품질(버그 등) 등 여러 가지가 있다. 이들은 모두 중요한 예측 대상이지만, 그중에서도 중요한 것은 개발 투자액(공수)과 개발 기간이다. 예측은 개발의 각 단계에서 실시되지만, 여기서는 주로 개발의 상류 공정에서 실시되는 예측을 다루기로 한다. 개발할 『소프트웨어의 양(크기)』과 개발 투자액(공수) 그리고 개발 기간이 어떤 함수 관계에 있다는 것은 쉽게 추정할 수 있다. 또, 소프트웨어의 양은 소프트웨어의 각종 특성을 평가할 때의 기준을 세우는데 도움이 된다. 그래서 일반적으로 소프트웨어의 양을 우선 추정/예측해서(즉, 소프트웨어의 규모를 견적), 이를 토대로 개발 투자액(공수)과 개발 기간을 예측하는 방법을 이용한다.

소프트웨어의 양을 나타내는 척도에도 여러 가지가 있지만 다음의 두가지가 대표적이다.

◉ 프로그램 행수([S]LOC : [Source] Lines Of Code의 약자)
◉ 소프트웨어의 기능량

많은 사람들은 프로그램 행수에 대해 막연한 이미지를 가지고 있으면서도 지금까지는 소프트웨어의 양의 척도로서 그다지 저항없이 받아들였다. 이것은 프로그램 행수가 소프트웨어의 속성 중에서는 보기 드물게 직접 『눈으로 볼 수 있는 양』으로서, 사람이 보거나 만져 봄으로써 사물을 이해한다는 특질에 잘 맞기 때문이다. 프로그램 행수가 어떤 면에서는 소프트웨어의 양을 나타내고 있는 것도 사실이다. 그러나, 최근에는 프로그램 행수는 예측을 위해 사용하는 소프트웨어의 양으로서는 적절하지 않다는 인식이 높아졌다.

소프트웨어 기능량은 소프트웨어가 가지고 있는 또는, 가지게 될 기능을 어떤 일정한 수단에 의해 정량화한 것이다. 소프트웨어 기능량의 대표적인 것으로, 여기서 소개하는 평션 포인트

† 일본 정보처리학회 1994년 4월호 게재 기사임.
†† 일본 NTT 소프트웨어연구소 근무

(function point : 기능 점수)가 있다. 그외에 DeMarco의 Bang척도 와 화면수에 의해 소프트웨어의 규모를 규정하는 방법 등도 이 분류에 속한다[1]. 프로그램 행수와는 달리 소프트웨어 기능량은 소프트웨어의 예측과 개발에 관한 중요한 판단 근거라는 인식이 높아지고 있다. 그러나 소프트웨어 기능량은 프로그램 행수와는 대조적으로 직접 눈으로 볼 수 없는 추상적인 양이다. 이 점이 소프트웨어 기능량이 소프트웨어에 있어서 중요한 척도임에도 불구하고 지금까지 그다지 보급되지 않은 원인 중의 하나라고 여겨진다.

그러나 현재 행해지고 있는 소프트웨어 규모 견적법은 소프트웨어 기능량을 사용하는 방향으로 나아가고 있다. 여기서는 소프트웨어 규모 견적의 대표적이고 안정된 방법인 펑션 포인트법에 대해 방법의 개요, 이점, 소프트웨어 개발 프로세스와의 관계, 이용 상황 등에 대해 설명하기로 한다.

2. 펑션 포인트법의 개요

펑션 포인트법은 미국 IBM의 A.J.Albrecht에 의해 1979년에 초판이 공표되었고, 1983년에는 거의 현재의 형태로 정리된 버전이 공표되었다[2, 3]. 그후 이것을 토대로 각종 펑션 포인트법이 제안되었지만 여기서는 Albrecht(1983년)판을 보다 개량한 IFPUG법을 토대로 해설을 하기로 한다.

2.1 펑션 포인트법의 시점

소프트웨어 개발이란 사용하는 측(유저)의 요구를 개발하는 측이 가진 실현 방법으로 변환하는 과정으로, 항상 유저 시점과 개발자 시점의 두가지 시점이 있다. 유저 시점이란 소프트웨어로 무엇을 할 수 있는가(기능)라는 것을 의미하고, 개발자 시점이란 소프트웨어를 어떻게 만들것인가하는 것을 의미한다. 펑션 포인트법에서는 소프트웨어를 유저 시점에서 볼 것을 요구한다. 이것이 펑션 포인트가 기능량이라고 불리는 이유로서, 펑션 포인트법을 이해하고, 실제로 적용하는데 있어서 중요한 개념이다. 이에 대해 LOC는 개발자측의 시점에서 본 양이다.

펑션 포인트가 무엇을 나타내는가를 구체적으로 이해하기 위해 펑션 포인트를 구하는 방법에 대해 설명하기로 한다.

2.2 펑션 포인트의 계산법

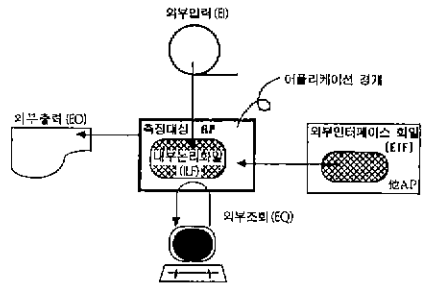
펑션 포인트는 다음의 절차에 의해 구할 수 있다.

step 1 : 측정 대상(소프트웨어 시스템)을 명확하게 한다(어플리케이션 경계)
step 2 : 펑션을 추출한다
step 3 : 펑션의 복잡도를 평가한다
step 4 : 평가된 복잡도를 합산한다
step 5 : 시스템의 특성을 평가한다
step 6 : 펑션 포인트를 구한다

상기 스텝은 논리적인 스텝으로서, 일반적으로 step 2와 step 3은 거의 동시에 행해진다. 다음에 각 단계를 보다 자세히 설명하기로 한다.

(1) 측정 대상(소프트웨어 시스템)을 명확하게 한다

측정한 값의 의미를 명확하게 하기 위해서는 측정할 대상이 어떤 것인지를 분명히 할 필요가 있다. 펑션 포인트법에서는 『유저 시점』에서 측정하는 대상을 분명히 할 것을 요구한다. 이 측정 대상을 둘러싸고 있는 가상적인 경계를 『어플리케이션 경계』라고 부른다.



(그림 1) 펑션 포인트법의 5가지 펑션

그래서, 펑션 포인트법에서는 소프트웨어 또는 소프트웨어 시스템을 어플리케이션이라고 부르는 경우가 많다.

(2) 펑션을 추출한다

어플리케이션 경계를 명확하게 한 후, 측정 대상인 소프트웨어의 요구 사양서와 기능 사양서로부터, 『유저 관점에서』 (그림 1)에 나타내는 다섯가지 펑션(기능)을 추출한다.

다섯가지 펑션의 추출은 어플이케이션내에서 취급되는 데이타의 성질에 주목해서 행해진다. 이것이 펑션 포인트법의 특징이다.

① 외부 입력 : 어플리케이션 경계를 넘어서 시스템에 데이타를 입력하는 기능을 말한다. 유저가 직접 입력하는 경우만이 아니라, 다른 어플리케이션이 작성한 데이타를 MT와 통신 회선을 통해서 입력하는 경우도 외부 입력에 포함된다.

② 외부 출력 : 어플리케이션 경계를 넘어서 시스템에서 데이타를 출력하는 기능을 말한다. 프린터에 출력되는 장표, 화면 출력, MT 등의 2차 기억에 출력되는 데이타가 이에 해당한다.

③ 외부 조회 : 사용자의 입력에 의해서 시스템 내에 있는 데이타를 출력하도록 입출력이 짝을 이루는 기능을 말한다.

④ 내부 논리 화일 : 어플리케이션 경계 안에서 유지되는 논리적으로 관련된 데이타의 집합 중에서 유저 시점에서 보았을 때 필요한 것을 취급하는 기능을 말한다. 여기서 유지란 내부 논리 화일의 데이타 생성, 사용, 추가, 갱신, 삭제 등을 가리킨다.

⑤ 외부 인터페이스 화일 : 어플리케이션 경계 밖에 있는 소프트웨어가 유지하는 논리적으로 관련된 데이타의 집합 중 유저측에서 필요한 것을 참조하는 기능을 말한다.

(3) 펑션의 복잡도를 평가한다

추출된 각 펑션의 『복잡도』를 구해서 그에 맞는 점수를 할당한다. 모든 펑션에 있어서 복잡도의 정도는 『높다』,『중간이다』,『낮다』는 3종류뿐이다. 복잡도의 정도는 펑션이 다루는 데이타 요소의 수와 다음의 ①, ②의 조합에 의해 측정된다.

① 입출력 기능에서 참조하는 화일수(외부입력, 외부출력, 외부참조의 경우)

② 화일이 포함하는 레코드의 종류수(내부논리 화일, 외부인터페이스 화일의 경우)

점수는 펑션과 복잡도의 정도의 조합으로 미리 정해져 있다. 〈표 1〉과 〈표 2〉에 외부 입력과 내부 논리 화일에 대한 복잡도 평가 및 점수를 할당하기 위한 표를 나타낸다.

〈표 1〉 복잡도의 평가(EI : 외부 입력)

데이타 요소수 / 참조화일수	1 ~ 4	5 ~ 15	16 ~
0, 1	저	저	중
2	저	중	고
3 ~	중	고	고

저 = 3　중 = 4　고 = 6

〈표 2〉 복잡도의 평가(ILF : 내부논리 화일)

데이타 요소수 / 레코드종류수	1 ~ 19	20 ~ 50	51 ~
1	저	저	중
2 ~ 5	저	중	고
6 ~	중	고	고

저 = 7　중 = 10　고 = 15

(4) 평가된 복잡도를 합한다

추출된 모든 펑션에 대한 복잡도의 평가값을 더한다. 얻어진 합계값을 미조정 펑션 포인트라고 부르기도 한다.

(5) 시스템의 특성을 평가한다

상기 (3)의 복잡도 평가만으로는 어플리케이션의 복잡도를 충분히 평가할 수 없는 경우가 있다.

그래서 〈표 3〉에 나타낸 14종류의 평가 항목에 대한 평가값에 의해 (4)에서 얻어진 복잡도의 평가값의 합계값을 조정한다. 이 14종류의 평가 항목을 시스템 특성이라고 부른다. 시스템 특성의 각 항목은 각각 0 ~ 5점의 6단계로 평가되고, 14항목 전체에 대한 평가값의 합계값을 구해, 시스템 특성의 평가값으로 한다.

〈표 3〉 시스템 특성

NO	분 류	특 성 항 목
1	알고리즘의 복잡도	
2	온라인처리	통신기능을 통한 데이타 수집의 복잡도
3		온라인 데이타 입력 비율
4		내부데이타베이스가 온라인 갱신되는 비율
5	분산처리	분산 처리의 고도성
6		복수 조직 사용에 따른 설계 제조에 대한 영향
7	처리성능	트랜잭션량이 설계에 주는 영향
8		응답시간, 처리량에 대한 요구의 엄밀성
9		고부하에 의한 설계 요구의 엄밀성
10	사용용이성,조작성	엔드유저의 사용용이성을 가능하게 하는 요구의 정도
11		엔드 유저가 기능 변경 가능하게 하는 요구의 정도
12		시스템의 운용의 용이에 대한 요구의 정도
13		시스템 이행및 인스톨의 용이성에 대한 요구의 정도
14	재이용가능하게 하는 설계의 요구 정도	

(6) 평션 포인트를 구한다

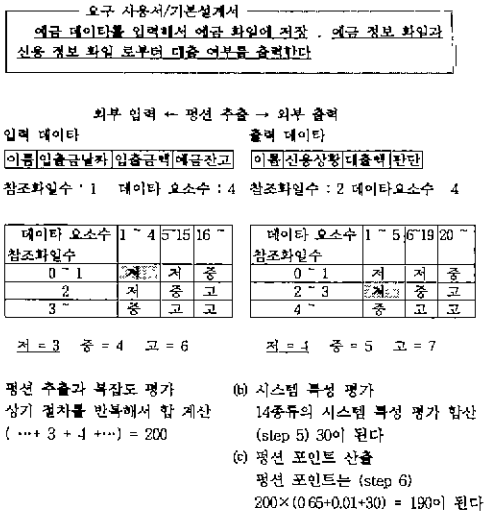
이상의 절차에 의해 얻어진 값을 다음 식에 대입해서 어플리케이션의 펑션 포인트를 구한다.

$$Fp = S_c \times (0.65 + 0.01 \times S_g)$$

여기서 F_q : 평션 포인트, S_c : 복잡도의 평가값의 합계 S_g : 시스템 특성의 평가값

일반적으로 이 식은 평션의 복잡도 평가에 의해 구해진 값을 ±35%의 범위에서 보정한다고 해석되고 있다. 실제, 표준적인 시스템에서는 상기 식의 괄호 안은 1에 가까운 값이 된다.

여기서 나타낸 식은 새로 개발할 어플리케이션의 평션 포인트를 구하는 식이다. 이외에 상기의 식을 토대로 기능 확장의 경우와 기능 확장후의 평션 포인트를 구하는 식이 있다.



(그림 2) 펑션 포인트 계산 예

이상에서 본 바와 같이 평션 포인트의 측정은 프로시져화되어 있으며, 그 결과 사람에 의존하지 않고, 재현성이 높으며 측정치로서 신뢰성이 높다.

(그림 2)는 간단한 평션 포인트의 계수예를 나타낸다.

2.3 펑션 포인트의 의미

다음에 기술한 문장은 Albrecht가 두개의 논문에서 설명한 것을 요약한 것으로써, 평션 포인트의 의미를 잘 나타내고 있다.

"평션 포인트는 프로그램의 복잡도와 관련되어 있고, 『어플리케이션의 모든 기능을 포함하는 외부 표현』과 비례 관계에 있는 양이다". 여기서 외부 표현이란 앞에서 설명한 『유저시점에서의 요구』에 대응된다. 그리고 Albrecht는 평션 포인트가 공수와 높은 상관 관계에 있고, 또 소프트웨어의 견적과 평가에 사용할 수 있다는 것을 나타냈다.

바꿔말하면 펑션 포인트는 『소프트웨어를 유저 시점에서 보았을 때 그 소프트웨어의 전체를 나타내고 있는 양이고, 소프트웨어 개발의 공수와 높은 상관 관계에 있는 실용적인 양이다』라는 것이다.

펑션 포인트의 복잡도 평가는 Albrecht가 『토의와 시행에 의해 결정』한 것이다[2, 3]. 시스템 특성에 대해서도 마찬가지이다. 이들 값과 항목은 유럽에서 약 10년에 걸친 사용 실적과 다음에 설명하는 『정도』가 좋다는 점을 볼 때, 적용 분야(후술)를 특정하면 타당한 것이라고 생각해도 좋다. 더우기, 회사 간의 비교 등을 위해서는 모든 이용자가 같은 항목과 값을 사용하고 있다는 것이 중요하다는 점에 주의할 필요가 있다.

2.4 펑션 포인트의 정도

펑션 포인트법의 정도에 대해서 살펴 보기로 한다. 여기서 정도란 재현성의 정도, 공수 견적의 정도 및 프로그램 행수 견적의 정도를 의미한다.

(1) 재현성의 정도

재현성이란 몇번을 하더라도 같은 결과가 되던가 아니면 누가 하더라도 같은 결과로 된다는 것을 나타내는 정도이다. 후자는 신뢰성이라고 불리기도 한다.

Kemerer는 신뢰성을 평가하기 위해, 27개 시스템을 54명의 계측자에게 독립적으로 계측시키는 실험을 했다[5]. 그 결과 각 계측자의 평션 포인트는 높은 상관을 가지고(상관 계수 : 0.8), 계측값의 오차가 적다(오차의 상대값의 메디안이 약 0.12)는 것을 나타내며, 평션 포인트법은 신뢰성이 높다고 주장하였다.

그리고, Kemerer등은 표준적인 해석과 달리 해석되는 가능과 그 평션 포인트 계측값에 대한 영향에 대해 연구를 하고 있다[6]. 그 결과, 1) 평션 포인트법의 규칙 해석은 통일되어 있다는 점, 2) 대다수 항목에 대한 해석의 차이는 평션 포인트 계측값에 대한 영향이 적다는 점, 3) 해석이 서로 다른 항목에 대해서도 규칙을 정비함으로써 해석을 하나로 할 수 있다는 점에서 평션 포인트법의 신뢰성은 높다고 한다.

저자등도 다른 각도에서 계수법의 문제점을 검토했으나 매뉴얼 기술의 충실과 엑스퍼트의 지원에 의해 계수는 정도높게 할 수 있다는 결과를 얻었다[7].

(2) 공수 견적의 정도

Albrecht는 24종류 어플리케이션의 데이타로부터 다음과 같은 공수와 평션 포인트의 관계식(회귀식)을 나타냈다.

$E_{mh}=54\times F_p-13390$ (상관 계수 : 0.935)

여기서 E_{mh} : 공수(人時) F_p : 평션 포인트

Kermerer는 15개 시스템에 대해 상기 Albrecht의 식에 의한 공수, COCOMO(Constructive COst MOdel의 약자)에 의한 공수, 및 SLIM에 의한 공수와 실공수를 비교했다[9, 10]. 그 결과 산출된 공수와 실공수의 비율은 평션 포인트의 경우는 1.2, LOC를 사용하는 COCOMO와 SLIM의 경우는 각각 5.6과 9.4이었다. 이 결과는 평션 포인트가 공수를 예측하기 위한 좋은 양이라는 것을 나타낸다. 단, 평션 포인트법에 상당히 유리한 결과로 된 것은 Kemerer과 Albrecht의 데이타의 특성이 유사했기 때문이라고 추정한다.

(3) 프로그램 행수 견적의 정도

후술하는 LOC의 문제점때문에 저자는 프로그램 행수와 평션 포인트와의 관계는 평션 포인트의 정도로서는 중요하지 않다고 생각한다. 그러나, 프로그램 행수로부터 평션 포인트로의 이행기에는 편리한 툴이 된다. 가령 Albrecht는 다음과 같은 회귀식을 만들었다.

$S_s=118.7\times F_p-6490$(상관 계수 : 0.854)

여기서 S_s : COBOL의 LOC F_p : 평션 포인트

2.5 평션 포인트법의 적용 대상 소프트웨어

평션 포인트법은 입출력과 화일을 기능으로 보고 있기 때문에, 이러한 처리가 지배적인 비지니스 소프트웨어(뱅킹, 사무 처리 계산 등)가 그 적용 영역이 된다. CPU 처리가 지배적인 과학기술 계산과 OS 등의 소프트웨어에 적용하기 위해, 소프프웨어의 특징(평션 포인트법의 평션)을 파악해서 소프트웨어의 규모를 정량화하는 평션 포인트법의 아이디어를 확장하려고 연구를 계속하고 있으나 아직 보고된 것은 그다지 많지 않다.

3. 평션 포인트와 프로그램 행수(LOC)

개발 투자액(공수)과 개발 기간의 예측은 일반적으로 소프트웨어의 양(또는 규모)을 토대로 행해진다는 것은 앞에서 설명했다. 따라서 공수와 기간에 대한 예측의 정도는 소프트웨어의 양의 정도(견적 정도)에 의존한다. 또, 이들 예측은 개발에 있어서 가능한 빠른 시기에 정확하게 하는 것이 중요하다.

소프트웨어의 양을 나타낸 것으로서 종래 많이 이용되어 온 프로그램 행수(LOC)는 개발의 초기 단계에서는 불분명하므로, 일반적으로 경험자의

경험과 감에 의해 추정된다. LOC 및 그들을 이용하는 방법은 다음과 같은 문제점을 가지고 있고, 소프트웨어 개발을 예측하기 위한 소프트웨어의 양 및 방법으로서는 그다지 타당한 것은 아니다.

① 사람에 의존하고(경험자의 『경험과 감』), 재현성이 부족하다(정해진 방법은 없다)

② LOC는 개발 결과로서 얻어지는 것으로, 개발 초기에 예측하는 것은 곤란하다(정도가 좋지 않다)

③ 개발 기술과 환경에 의존하고, 개발할 소프트웨어의 사양에 고유한 양을 나타내지 않는다

이에 대해 평션 포인트 및 펑션 포인트법은 다음과 같은 이점이 있고, 소프트웨어 개발을 예측하기 위한 소프트웨어의 양 및 방법으로서 뛰어나다. 더우기, 소프트웨어의 각종 속성을 평가하기위한 기준량으로서 이용할 수도 있다.

(1) 양으로서 신뢰할 수 있다.

계측 과정이 프로시저화되어 있기 때문에, 계측 과정을 기록해 두는 것과 이후의 산출 과정을 재현하는 것이 용이하다. 이 성질은 펑션 포인트의 신뢰성을 높히는데 도움이 된다.

또, 펑션 포인트는 요구 사양서와 기능 사양서로부터 점해진 절차에 따라 펑션을 추출, 평가함으로써 『측정하는』양이다. 『측정한다』는 것은 『예측』과는 달리 정도를 필요한 만큼 올릴 수가 있다. 보다 좋은 것은 펑션은 개발의 초기 단계에서는 비교적 쉽게 추출할 수 있다. 특히, 펑션 포인트법에서 중요한 의미를 가지는 입출력은 초기 단계에서도 명확하게 식별되어 있는 경우가 많다. 이때문에 개발 초기 단계에서 높은 정도로 소프트웨어의 양을 측정할 수 있다.

(2) 개발의 초기 단계에 이용하는 양으로서 적절하다.

펑션 포인트는 유저 시점에서의 양이다. 유저에게 있어서 소프트웨어는 기능이 중요하고, 어떻게 만들어지는가(이것이 프로그램 행수에 대응된다고도 할 수 있다)는 부차적인 경우가 많다. 개발 초기 단계의 견적은 유저와의 절충에 사용되는 경우가 많다. 유저에게 있어서는 이해하기 쉽고 또, 개발자에게 있어서는 유저와의 공통적인 인식을 가질 수 있는 유저 시점에서의 양을 토대로 한 견적은 중요하다.

(3) 개발 기술과 환경에 의존하지 않고, 개발할 소프트웨어의 사양에 고유한 양을 나타낸다

펑션 포인트의 개발 기술과 환경 및 언어(이들을 개발 수단이라고 부르기로 한다)에 대한 의존도가 매우 낮다는 것은 펑션 포인트의 계측 방법으로부터 자명하다. 이것은 개발 수단이 소프트웨어 개발에 주는 영향을 펑션 포인트에 의해 상세하게 분석할 수 있다는 것을 나타낸다. 이 때문에 개발 공수, 기간을 예측하는 경우에는 개발 수단의 영향을 예측에 제대로 반영할 수 있고, 보다 정도가 높은 예측을 할 수 있다는 것을 의미한다.

〈표 4〉에 이상을 정리해서 LOC와 펑션 포인트의 비교를 나타냈다.

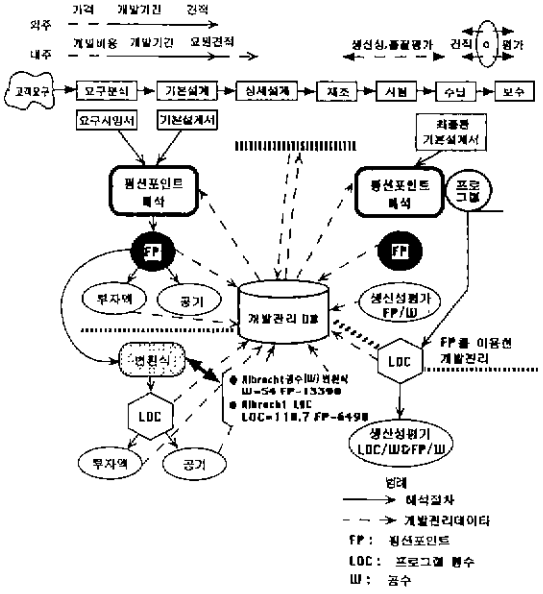
〈표 4〉 LOC와 펑션 포인트의 비교

개발 공수/기간 견적에 필요한 속성(특히 상류공정 견적)		LOC		펑션 포인트	
소프트웨어 양으로서의 신뢰성	산출법	낮다	경험과 감	높다	정해진 절차
	재현성		부족하다		뛰어나다
	획득법		예측		계측
개발 초기에서의 적용성	정도높게 획득할 수 있을 것	불가능		가능	
	유저와의 교섭에 사용할 수 있을 것	사용하기 어렵다(소프트웨어 작성 과정에 관계한 암으로서, 유저는 이해하기 어렵다)		사용하기 쉽다(소프트웨어 기능과 직결된 양으로 유저가 이해하기 쉽다)	
소프트웨어 사양에 고유한 것		고유하지 않다(개발기술,환경, 언어에 매우 의존)		고유하다(개발기술,환경, 언어에 의존하지 않는다)	

4. 소프트웨어 개발 프로세스와 펑션 포인트

펑션 포인트는 소프트웨어의 기본적인 속성인 『양』을 나타내고 있다. 이때문에, 펑션 포인트는 소프트웨어 개발의 여러 면에서 사용 가능하다. 단, 펑션 포인트는 소프트웨어의 작성 과정을 보

지 않기 때문에, 작성 과정을 관리하는데는 적합하지 않다고 생각한다. 따라서 펑션 포인트(법)을 소프트웨어 개발에서 적용하는 것은 1) 견적과 2) 평가가 대표적일 것이다. (그림 3)에 소프트웨어 개발 프로세스와 펑션 포인트 이용법의 관계를 나타낸다. 펑션 포인트를 이용하면 소프트웨어 개발의 여러 모양을 분석할 수가 있다. 그를 위해서는 펑션 포인트와 함께 개발에 관한 여러 데이타를 축적해 두는 것이 중요하다. 또, 여기서는 설명하기 쉽다는 점에서 개발 모델로서 워터폴 모델(waterfall model)을 가정하고 있는데 펑션 포인트법 자체는 모델과는 독립적이고, 프로토타이핑 등의 모델에도 적용할 수 있다.



(그림 3) 소프트웨어 개발 프로세스와 펑션 포인트의 이용법

(1) 견적

요구 분석 공정과 기본 설계 공정 사이에서 작성된 도큐멘트로부터 펑션 포인트를 계측한다. 계측한 펑션 포인트와 과거의 개발 데이타로부터 해당 소프트웨어 개발의 코스트와 기간을 견적한다.

정도가 그다지 문제로 되지 않으면 개발 계획 단계에서부터 사용할 수 있다. 이 때는 펑션의 복잡도, 입출력, 내부 논리 화일의 종류 등은 과거의 데이타로부터 가정한다. 이들 견적을 위해 토대로 사용한 데이타도 나중에 실적 데이타 등으로부터 검증할 수 있다.

펑션 포인트의 정도는 요구 사양 분석 또는 기능 설계의 정도라는 점에 주의할 필요가 있다. 계측 대상이 되는 것이 제대로 되어 있지 않으면 아무리 정밀하게 계측해도 무의미하다. 펑션 포인트법을 도입하는데 있어서, 『사양은 변경되는 것』이라는 종래 소프트웨어 개발의 『상식』을 되새겨 볼 필요가 있다. 『Garbage-in Garbage-out』라는 말이 있는데 펑션 포인트법을 적용할 때에도 명심해야 한다.

(2) 평가

시험 단계에서 납품 단계에 걸쳐 펑션 포인트를 이용해서 개발한 소프트웨어를 평가할 수가 있다. 그때 필요하다면 펑션 포인트를 재계측한다. 재계측은 개발한 시스템의 최종적인 펑션 포인트를 결정하기 위해서이다. 펑션 포인트를 사용한 지표의 예에는 다음과 같은 것이 있다.

① 사양변경률=최종 펑션 포인트/초기 펑션 포인트
② 생산성 1=프로그램 행수/펑션 포인트
③ 생산성 2=투입 코스트/펑션 포인트
④ 생산성 3=도큐멘트 매수/펑션 포인트
⑤ 품질=추출 버그수/펑션 포인트

펑션 포인트는 적어도 종래의 양의 지표이었던 프로그램 행수를 대신해서 보다 정도가 높은 지표를 제공한다. 또 상기의 사양 변경율과 같은 새로운 지표를 제공할 수도 있다. 펑션 포인트의 이러한 성질이 소프트웨어 개발의 정량화를 할 수 있게 한다고 생각한다.

5. 펑션 포인트법의 이용 상황

펑션 포인트법의 이용 상황 및 표준화 상황에 대해 간단히 설명한다.

(1) 일본의 이용 상황

일본에서는 평션 포인트법은 그다지 이용되지 않고 있다. 그러나 몇몇 큰 회사는 이미 평션 포인트법을 실제 개발에 사용하고 있다. 또, 많은 소프트하우스가 관심을 가지고 있으며, 자기 회사내에서 많은 연구를 하고 있다.

평션 포인트법이 보급되지 않은 원인은 많이 있겠지만, 그 도입법에도 문제가 있다. 齊藤등은 평션 포인트의 도입을 저해하는 요인을 고찰해서 톱다운적인 도입, 다른 방법과의 병용, 방법의 선택, 필요한 도큐멘트를 제안하고 있다[11].

(2) 기타 이용 상황

세계에서 가장 평션 포인트법이 많이 보급되어 있는 나라는 미국으로서, 이것을 지탱하는 것이 IFPUG이다. IFPUG는 미국에 본거지를 둔 평션 포인트법의 유저 단체로서, 현시점에서 de facto 한 평션 포인트 표준화 단체이다[12]. 회원은 금융, 공공 사업, 소프트하우스, 컴퓨터메이커 등 많지만 저자의 인상으로는 통신 관계의 회사가 많이 가입하고 있다. 또, 오스트레일리아, 캐나다, 남아프리카, 영국, 프랑스, 네델란드, 독일에 지부를 두고 있다.

미국에서 실제로 어느 정도 평션 포인트법이 사용되고 있는지는 알 수 없지만, 작년 가을 IFPUG의 유저 회의에서 보고된 다음의 AT&T의 데이타가 참고로 될 것이다[13].

- 10년 이상 사용해서, 계측한 평션 포인트수가 800,000이 된다.
- 600의 프로젝트에 적용했다
- 계측에는 60人年이 투입되었다

유럽도 평션 포인트법의 사용에 비교적 열심이다. 早稲田대학의 東등이 실시한 소프트웨어 관리와 매트릭스의 앙케이트 결과에 의하면 회답한 유럽 기업의 25%이상이 생산성의 지표로서 평션 포인트/단위 시간을 사용하고 있다고 한다. 이것은 LOC/단위 시간에 이어서 상당히 많이 사용되고 있는 지표이다[14].

(3) 표준화

평션 포인트법을 ISO의 새로운 검토 항목(NWI)으로 한다는 것이 1993 6월의 국제회의에서 제안되었고, 12월 투표에서 정식 검토 항목으로 인정되었다[15]. 앞으로 검토가 활성화되리라고 생각한다. 이 제안은 IFPUG파인 네델란드, 오스트레일리아가 주체가 되어 행해졌으며, 표준은 IFPUG베이스가 될 것으로 예상된다.

6. 평션 포인트법의 변형

평션 포인트법에는 여러 변형이 있는데 적용 영역을 확장할 것인지 어떤지로 크게 나눌 수 있다.

(1) 적용 영역을 확장하지 않는 것

이 분류에 속하는 de facto standard적인 방법이 IFPUG법이다. IFPUG법은 복잡도 평가의 객관화와 룰의 정밀화/적성화 등의 변경은 하고 있지만, 기본적으로는 Albrecht의 수법을 그대로 이어받은 것이다.

IFPUG법 이외의 방법으로서는 일본에서는 JISA, IPA, COSDES에 의한 방법이 있다. 기타 선진 외국에서는 C. Jones 등에 의한 SPR 법, C. R. Symons에 의한 MARK II 법 등이 있다[16, 17,18, 19, 20].

(2) 적용 영역을 확장하는 것

이에 속하는 방법의 수는 많지 않지만 과학 기술 계산과 리얼타임 소프트웨어에의 적용을 목적으로해서 C.Jones가 개발한 피처 포인트법[19], 프로세스제어용 소프트웨어에 적용할 목적으로 Mukhopadhyay등에 의해 개발된 방법 등이 있다[21].

7. 향후 과제

평션 포인트법을 유효한 척도로 하기 위해서는 소프트웨어 개발에 관한 각종 데이타를 축적함과 동시에 그 분석의 노우하우도 축적할 필요가 있다. 이들은 평션 포인트법을 이용해 가는 개발 환경의 과제이다.

평션 포인트법의 기술적인 연구 과제로서는 다음과 같은 것이 있다.

(1) 펑션 포인트법에 적합한 사양 기술법

현재 많이 사용되고 있는 자연 언어와 도표류에 의한 사양 기술에서는 펑션 포인트와 관련 화일이 명확하게 판정할 수 있는 기술법을 고안할 필요가 있다.

앞으로 보급이 진행될 것으로 생각되는 구조화 분석 설계법에 대해서 IFPUG는 가까운 장래에 계수 규약을 이에 적합하도록 개정할 예정이다. 그러나, 이것은 완성을 의미하는 것이 아니라 향후 적용을 해 나가서, 개발 현장에서 발생하는 구체적인 문제점을 분석해서, 기법에 피드백을 할 필요가 있다.

(2) 적용 분야의 확장

데이타가 적고 CPU를 많이 사용하는 소프트웨어에도 기능량에 해당하는 개념을 적용해서 그 양을 제대로 나타낼 필요가 있다. 이 어프로치는 이미 시작되었지만 평가는 확정되지 않았고, 이 분야의 연구는 극히 초보 단계라고 말할 수 있다.

(3) 자동 계측

펑션 포인트를 보다 정확하고 보다 쉽게 계측하기 위해서는 자동 계측 툴이 필요하다. CASE 툴과 조합해서 펑션 포인트 계측의 자동화를 꾀하려는 툴도 있는데 원래의 자동화와는 상당히 거리가 있다. 자동 계측율의 향상을 위한 연구가 필요하다.

8. 결 론

펑션 포인트법에 대해서 방법의 개략과 펑션 포인트의 의미, 종래 기법과의 비교, 적용법, 동향 등에 대해 설명했다. 펑션 포인트법은 100%의 완성도를 가진 방법은 아니다. 그러나 종래의 프로그램 행수를 토대로 한 기법에 비해 상당히 발전된 방법이라고 생각한다.

좋은 매뉴얼과 기술을 쌓음으로 인해, 펑션 포인트법을 정도가 높고 빨리 측정할 수 있다. 큰 프로젝트일 경우, 펑션 포인트의 계측 공수는 거의 무시할 수 있는 값으로서, 시험적인 사용을 통해 감각을 잡는 것이 중요하다.

참 고 문 헌

1. DeMarco, T : Controlling Software Project : Management, Measurement & Estimation, Yordon Press(1982).
2. Albrecht, A.J. : Measuring Application Develop-ment Productivity, Proc. Joint SHARE/GUIDE/IBM Appl. Develop. Symposium, pp. 83-92(1979).
3. Albrecht, A. and Gaffney, J. Jr. : Software Function, Source Lines of Code, and Development Effort Prediction : A Software Science Validation, IEEE Tr. on SE, Vol. 9, No. 6, pp. 639-648(1983).
4. IFPUG : IFPUG Function Point Counting Practice Manual, Release 3.4(1992).
5. Kemerer, C.F. : Reliability of Function Points Measurement, CACM, Vol. 36, No. 2, pp. 85-97(1993).
6. Kemerer, C.F. : Improving the Reliability of Function Point Measurement : An Empirical Study, IEEE Tr. on Software Engineering, Vol. 18, No. 11. pp.1011-1024(1992).
7. 西山, 齊藤, 古山 : ファンクションポイント計數上の問題點に關する分析, 1994年電子情報通信學術春季大會, D-95.
8. Boehm, B. : Software Engineering Economics, Prentice-Hall(1981).
9. Putnam. L.H. : A General Empirical Solution to the Macro Software Sizing and Estimation Problem, IEEE Tr. on Software Engineering, Vol.4, pp. 345-381(1978).
10. Kemerer, C.F. : An Empirical Validation of Software Cost Estimation Models, CACM, Vol. 30, No. 5, pp. 416-429(1987).
11. 齊藤, 西山, 古山 : FP法導入に關する一提案, 情報處理學會第48回全國大會, 4K-10.

12. IFPUG : New Member Orientation, IFPUG 1993 Fall Conference Proceeding.
13. Lubashevsky, A. : Living on the Edge at AT & T, IFPUG 1993 Fall Conference Proceeding.
14. Azuma, M. and Mole, D. : Softare Management Practice and Metrics : EC and Japan-Some Result of Questionaire Survey, Proc. 2nd International Conference on Achieving Quality Software, pp.57-71(1883).
15. ISO/IEC JTC 1/N 2629 Attachment N 1067 A : Function Point Analysis(1993).
16. 情報サ-ビス産業協會(JISA) : 情報處理工學に關する調査研究 ソフトウェアコストモデルの定量的評價
17. 情報處理振興事業協會(IPA) : ソフトウェアの規模見積り手法の調査研究最終報告, 平成2年3月.
18. ソフトウェア開發見積りシステム技術委員會(COSDES) : COSDES版FPAミニセミナ-資料(平成5年10月)
19. Jones, C. 著 : 鶴保征城, 富野壽監譯 : ソフトウェア開發の定量化手法, 共立出版(1993).
20. Symons, C. : Software Sizing and Estimation, John Wiley & Sons(1991).
21. Mukhopadhyay, T. : Software Effort Models for Early Estimation of Process Control Applications, IEEE Tr. on Software Engineering, Vol. 18, No. 10, pp. 915-924(1992).
22. Dreger, B. : Function Point Analysys, Prentice-Hall(1991).

◈ 알 림 ◈

당 학회에 대한 건의사항 또는 학회지와 논문지에 대한 건의사항 등이 있으시면 당학회 발전을 위해 많은 의견주시기 바랍니다.