|  |
| --- |
| **소프트웨어 규모의 산정** |
| 학과: 전기컴퓨터공학부 정보컴퓨터공학전공 학번: 201524582 성명: 정희석 |

**1. 소프트웨어 규모 산정**

1.1 소프트웨어 규모 산정의 필요성

소프트웨어의 개발에서 소프트웨어 규모의 산정은 매우 중요하다. 소프트웨어 개발은 주로 기획자, 개발자, 디자이너로 이루어지는 경우가 많으며 디자이너는 개발 상황이 눈으로 확인이 가능하여 기획자의 입장에서는 개발 진척도를 알 수 있으나, 개발자가 어느 정도 개발을 진행했는 지에 대한 진척도가 잘 안 보이는 특성, 다시 말해, 비가시성이 높기 때문에 이 규모를 산정함으로써 해당 개발 프로젝트의 예산, 필요 인력, 소요 기간 등을 예측 및 계산을 통해 개발 계획을 세우고 효율적인 개발 인력 투입을 통해 불필요한 자원낭비를 방지할 수 있다.

1.2 소프트웨어 규모 산정 방법

소프트웨어 개발 프로젝트의 규모를 산정하는 방법에는 큰 카테고리로 2가지 방법이 존재한다.

첫 번째는 LOC(Line Of Code)방식이 존재한다. 해당 방식은 개발 프로젝트에 작성된 소스코드들의 Line의 개수를 세는 방식이다.

두 번째는 FP(Function Point)방식이 존재한다. 해당 방식은 소프트웨어가 가지고 있는 기능의 개수를 세는 방식이다.

1.2.1 LOC(Line Of Code)산정 방법

- 개요: 소프트웨어의 소스코드의 행수를 의미한다. 주로 소스코드의 행수를 뜻하기 위해 SLOC(Source Lines Of Code)로 칭해진다. 텍스트 파일로서의 행수를 Physical LOC(물리LOC)라 하고 PLOC에서 개행 문자와 공백으로 이루어진 행, 주석으로 이루어진 행을 제외한 실제 명령의 행수를 Logical LOC(논리LOC)라 한다. Physical LOC를 그대로 쓰는 일은 매우 적으며 Logical LOC를 주로 사용한다.[[1]](#endnote-1)

- 방법: 예시로 두개의 소스코드를 비교하여 PLOC와 LLOC를 비교해보자.

예시 1)

A) **for** (i = 0; i < 100; i++) printf("hello"); */\* How many lines of code is this? \*/*

B)

*/\* How many lines of code is this? \*/*

for (i = 0; i < 100; i++)

{

printf("hello");

}

이 두 코드의 LOC를 비교해보면

[표 1]

|  |  |
| --- | --- |
| A code | B code |
| Physical LOC | 4 Physical LOC |
| 2 Logical LOC | 2 Logical LOC |
| 1 Command Line | 1 Command Line |

같은 기능을 하는 두 코드가 글자의 배열에 따라 Physical LOC가 달라지고 실제로 동작하는 부분을 세는 Logical LOC는 같으므로 PLOC보다 LLOC가 더 효율적임을 알 수 있다[[2]](#endnote-2)

- 장점:

1. 산정과정을 자동화하기 쉽다.

-> 물리적 LOC는 단순히 소스코드의 Line수를 세기만 하면 되고, 논리적 LOC는 주석처리, 공백, 명령이 되지 않는 글자로 이루어진 Line을 세기만 하면 되기 때문이다.

2. 직관적인 계측을 보여준다.

-> 산정 결과가 숫자로 나타나며 결과를 봤을 때 이해하기 쉽기 때문이다.

- 단점:

1. 사용하는 언어에 따라 산정 결과가 달라질 수 있다.

-> 예를 들면 C언어와 COBOL언어의 문법적 차이를 예로 들 수 있다.

2. 작성하는 프로그래머의 경험, 기술, 사용 언어에 따라 산정결과가 달라질 수 있다.

-> 프로그래머의 Coding Style, 주석 처리 방식 등에 의해 소스코드의 길이가 다르기 때문이다.

3. 이 산정 방식은 프로젝트 계획단계에서 실제로 구현되지 않으면 측정할 수 없다.

-> 실재하는 소스코드의 길이를 재는 것이기 때문에 예측하여 측정할 수 없다.

1.2.2 FP(Function Point)산정 방법

- 개요: Function Point는 기능 점수라 불리며 사용자의 관점에서 소프트웨어 규모를 산정하는 방법이며, 주로 논리적 설계를 기초로 하여 소프트웨어가 사용자에게 제공하는 기능의 수를 수치로 정량화 하고 소프트웨어의 규모를 산정하는 방식이다. 국내 소프트웨어사업대가기준은 국제 기능 점수 사용자 그룹(IFPUG)에서 제안한 모형을 기반으로 산정한다.[[3]](#endnote-3)

- 방법: 데이터기능점수와 트랜잭션기능점수를 아래 그림1과 같이 산정한다.



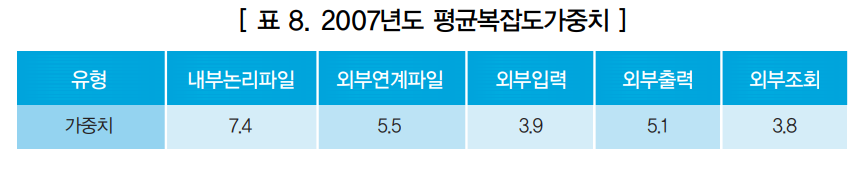
[그림1. 데이터 기능과 트랜잭션 기능 및 기능점수 산정][[4]](#endnote-4)

위의 데이터기능점수와 트랜잭션기능점수를 합하면 보정 전 기능점수가 되며 IFPUG에서 지정한 보정계수를 곱하면 기능점수를 구할 수 있다.

예시로 2007년 평균 복잡도 가중치는 아래 표2와 같다.

[표2]

[[5]](#endnote-5)



- 장점:

1. 사용하는 언어에 상관없이 산정된다.

-> 기능적으로만 따지기 때문에 어떤 언어를 사용하던지 상관이 없다.

2. 실제 구현하지 않고 추정하여 산정할 수 있음.

-> 계획단계에서 산정이 가능.

- 단점:

1. 계산하는데 시간이 오래 걸린다.

-> 내,외부 논리파일 기능점수, 외부 입,출력,조회 기능점수, 복잡도 가중치를 계산해야 되기 때문이다.

2. 가중치 계산을 위해 양질의 계산 결과 기록들을 요구한다.

-> 각 복잡도 가중치와 보정계수가 기존의 계산결과 들로부터 도출되기 때문이다.

3. 추정 결과 수치 사용으로 인한 실제와 다른 결과가 도출될 가능성이 있다.[[6]](#endnote-6)

-> 개발 단계에서 추가되는 기능, 삭제되는 기능, 사양변경 등의 다양한 요인이 존재하기 때문이다.

1. <https://ja.wikipedia.org/wiki/LOC> [↑](#endnote-ref-1)
2. <https://en.wikipedia.org/wiki/Source_lines_of_code> [↑](#endnote-ref-2)
3. 송기호, 기능점수(Function Point)산정 및 활용방안, 한국소프트웨어진흥원 정책연구센터, 73P, 2007 [↑](#endnote-ref-3)
4. 송기호, 기능점수(Function Point)산정 및 활용방안, 한국소프트웨어진흥원 정책연구센터, 73P, 2007 [↑](#endnote-ref-4)
5. 송기호, 기능점수(Function Point)산정 및 활용방안, 한국소프트웨어진흥원 정책연구센터, 76P, 2007 [↑](#endnote-ref-5)
6. Tucker Moore, Introduction and Basic Overview as an Alternative to SLOC-based Estimation, TASC – ASOU, 2011 [↑](#endnote-ref-6)