소프트웨어 복잡성 측정 시스템의 설계 및 구현

이 하 용† 이 용 근† 박 정 호†† 양 해 술

요 약

최근 소프트웨어에 대한 사용자의 이해가 높아짐에 따라 소프트웨어 개발자는 사용자의 요구를 만족시키기 위해 더 많은 노력을 하게 되었다. 따라서 소프트웨어는 규모가 방대해지고 복잡해졌다. 그로인해 소프트웨어의 개발 및 유지보수 비용은 증가되었고 개발자의 대다수는 유지보수에 투입되어 새로운 소프트웨어의 개발에 적체현상을 가져오게 되었다. 유지보수성이 좋은 소프트웨어는 하나의 모듈에 하나의 기능을 가지며 읽기 쉽고 복잡하지 않은 구조를 가져야 한다. 본 논문에서는 소프트웨어의 복잡성을 효과적으로 관리하기위해 소스프로그램을 입력으로 하여 프로덕트 메트릭스를 측정하고 요인항목들의 값을 산출하는 시스템을 설계하고 구현하였다.

The Design and Implementation of a Software Complexity Measurement System

Ha Yong Lee†, Ryong Geun Rhee†, Jung Ho Park††
and Hae Sool Yang

ABSTRACT

Recently, as users' understanding about software is raised, software developers devoted all their energy to satisfy users' needs. Accordingly, software is getting increase in volume and becoming complicated. As accoount of it, development and maintenance costs of software have been increased, and a large number of developers were projected in maintenance and development of new software. Software with good maintenability has a function in a module and is easy to read and simple. In this paper, for effective management of software complexity, I designed and implemented the system which accepts source program by input and measures product metrics and produce measurement value of factor items.

1. 서 론

오늘날 소프트웨어 라이프사이클에서 유지보수가 차지하는 비용이 전체 비용의 80%를 육박하게 됨으로써 소프트웨어의 유지보수가 개발 이상의 중요한 문제로 대두되었다[9]. 그로 인해 가능한한 최소의 비용으로 소프트웨어의 유지보수를 달성하는 일이 소프트웨어 개발자들에게 중요한 문제로 부각되었다.

유지보수성이 좋은 소프트웨어는 소스 프로그램의 각각의 모듈이 유지하고 있는 기능이 작고, 하나의 모듈당 하나의 기능을 가지고 있는 것이 바람직하며, 작성된 소스 프로그램이 시각적으로 읽기 쉬워서 프로그램을 쉽게 이해할 수 있어야 하고, 소스 프로그램의 논리가 단순하고 명쾌하여 논리적인 흐름을 이해하기 쉬워야 한다[6]. 본 논문에서는 소프트웨어의 복잡성을 효율적으로 관리하기 위하여 소스 프로그램을 입력으로 하여 소프트웨어의 특질을 측정하기 위한 기반이 되는 프로덕트(product : 제품으로서의 소프트웨어 자체) 메트릭스를 구하고 작성된 메트릭스를 기반으로 하여 복잡성 측정 모델에 정의된 요인항목에따라 필요한 프로덕트 메트릭스를 선택하

† 정 회 원 : 강원대학교 전자계산학과 박사과정
†† 종신회원 : 선문대학교 전자계산학과 조교수
논문접수 : 1995년 1월 24일, 심사완료 : 1995년 4월 10일

여 요인항목들의 측정값을 산출하는 시스템을 설계하고 구현하였다. 복잡성 모델을 구현함에 있어서 복수의 복잡성 측정 모델을 동일 대상(소프트웨어)에 적용하여 하나의 시스템 내에서 복수의 평가 관점을 도입하여 평가할 수 있도록 하였다. 이러한 측정 결과를 피드백 함으로써 소프트웨어의 복잡성을 줄이고 품질을 향상시킬 수 있다.

본 연구의 구성은 제2장에서 소프트웨어의 품질평가에 대한 개념과 평정수준의 설정에 관해 살펴보고, 3장에서는 소프트웨어의 품질평가를 위해 구현 대상으로 한 복잡성 모델에 관해 기술하고, 4장에서는 복잡성 모델들의 결과값을 산출하기 위해 소스 프로그램으로부터 추출하는 프로덕트 메트릭스에 관해 살펴보고, 5장에서는 소프트웨어 복잡성 측정 시스템의 구성 및 구현 결과를 논하고, 마지막 6장에서 결론 및 향후 연구과제를 제시한다.

2. 소프트웨어의 품질평가와 평정수준의 결정

2.1 소프트웨어 품질과 품질평가

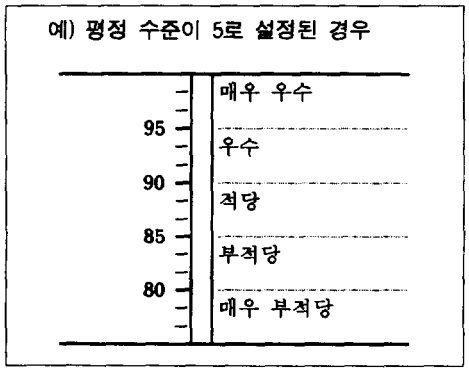
소프트웨어의 품질(SQ : Software Quality)은 소프트웨어의 사용 목적을 충족시켜 주는 소프트웨어 제품의 여러 속성의 정도를 나타내는 것으로써, 사용자의 이용 편의는 물론, 사용자 요구의 만족 정도를 표현할 수 있으며, 운영상의 결함이나 오류가 없음을 나타내는 척도이다[4]. 소프트웨어가 갖추어야 하는 적절한 특성으로서 얼마나 사용하기 쉬운지를 나타내는 사용성, 나중에 보수하기가 얼마나 쉬운가를 나타내는 유지보수성, 그리고 다른 환경으로의 이식성 등이 매우 중요한 의미를 갖게 된다[7].

소프트웨어 품질평가는 소프트웨어 개발 종료 후, 개발 과정에서 생성된 생산물과 최종 생산물인 소프트웨어를 평가함으로써 생명주기 전과정에서 발생된 오류를 검출할 수 있고, 이를 바탕으로 소프트웨어가 실제 요구자의 요구를 어느 정도 충족시키고 있는지를 정량적으로 평가함으로써 소프트웨어의 신뢰성을 측정평가하는 평가방법론이다.

2.2 품질평가에 대한 평정 수준의 결정

소프트웨어 품질평가에 의한 결과 자체는 큰 의미를 지니지 못하므로 평정 과정을 거쳐 품질 수준이 결정되어야 한다. 평정이란 소프트웨어에 대한 측정값을 설정된 기준과 비교하여 품질이 어느 수준에 이르고 있는지를 결정하는 것으로 측정값이 정량적이고 객관적이며 평정 수준이 표준화되어 있다면 품질 수준을 매우 정확하게 평가할 수 있다.

(그림 1)은 소프트웨어에 대한 품질 평가가 이루어진 후에 평가 결과에 대해 평정 수준을 5로 설정하여 소프트웨어의 품질수준을 나타낸 것이다.



(그림 1) 평정수준 설정 예
(Fig. 1) Example of evaluation level setup

3. 소프트웨어의 품질평가를 위한 복잡성 모델

3.1 Halstead의 복잡성 척도[1, 2, 3]

소스프로그램을 통해 측정 가능한 특징들에 다음과 같은 것들이 있다.

n_1 = 별개의 오퍼레이터의 수
n_2 = 별개의 오퍼랜드의 수
N_1 = 오퍼레이터의 총 출현 횟수
N_2 = 오퍼랜드의 총 출현 횟수

Halstead의 복잡성 척도는 위에 제시된 4가지 항목들을 조합하여 측정에 이용하는 것으로 복잡성 메트릭스 중 매우 신뢰받을 만한 측정으로 인

정되고 있으며, 적용과 검증을 통해 정확성이 뛰어나다고 판정되었다[5]. Halstead의 복잡성 척도에는 다음과 같은 것들이 있다.

어휘 사이즈 : n_1+n_2
프로그램 길이 : N_1+N_2
프로그램 볼륨 : $(N_1+N_2)\ \log_2(n_1+n_2)$

3.2 McCabe의 복잡성 척도[1, 2, 3]

제어흐름 그래프상에서의 McCabe의 사이클로매틱수(Cyclomatic number)로 복잡성을 구한다.

$C(G)=e-n+2P$
C : 사이클로매틱수　　e : 에지의 수
n : 노드의 수　　P : 연결성분의 수

단, P는 단일 모듈에 대해 1의 값을 갖는다.

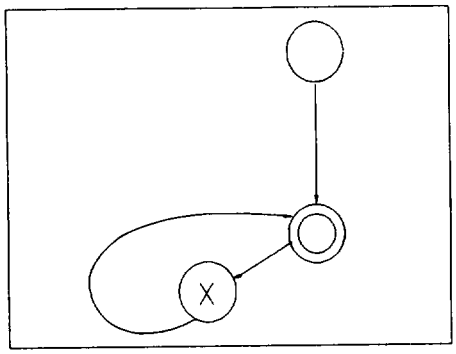
3.3 D 척도[9]

(1) 정의

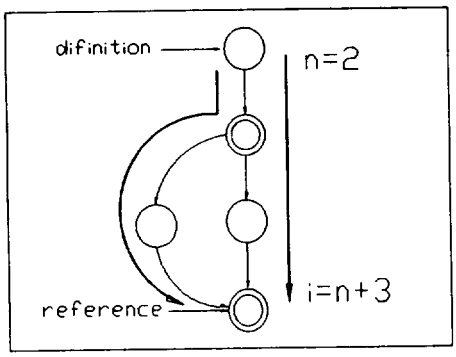
어떤 하나의 변수 x에 대하여 G의 출구에 대한 출력 대입수를 최대로 하는 각 구조에의 대입을 배치하고, 그 G에서의 x의 대입-참조 체인의 최대수를 척도 D로 정의한다. 구조화 모듈 M의 복잡성 D는 M으로부터 변환된 제어 데이타 그래프 G로 평가한다.

(2) 척도의 제약 조건

1) 제어데이타 그래프 G내의 각 블록에서는 단지 1회의 대입이 있고, 조건부에서는 없다.



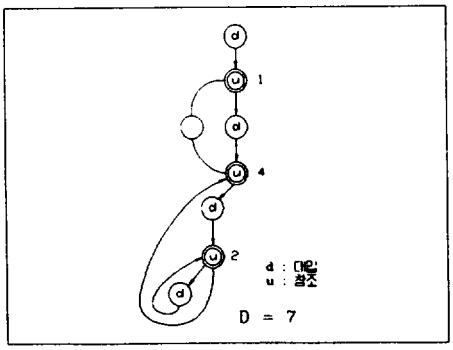
(그림 2) 그래프 G상의 출력 대입
(Fig. 2) Output definition in graph G



(그림 3) 그래프 G상의 대입 참조 체인
(Fig. 3) Definition-reference chain in graph G

2) 제어 구조의 조건부에서는 단지 1회의 참조가 있고, 블록에서는 없다.

(3) D 척도의 값을 구하는 예



그림에서 보면 참조된 위치에서 대입-참조 체인의 수를 표시하고 있다. 첫번째 참조에서는 대입-참조 체인의 수는 1개이며, 두번째 참조에서는 최초의 대입에 대한 참조와 새로운 대입에 대한 참조 및 루프의 형태로 나타나는 두개의 대입에 대한 참조를 포함하여 모두 4개의 대입-참조 체인이 형성된다. 세번째 참조에서 나타나는 2개의 대입-참조 체인을 모두 합하여 계산하면 D 척도의 값은 7이 됨을 알 수 있다.

4. 소프트웨어 품질평가 메트릭스

4.1 메트릭스의 유형

(1) 프로세스 메트릭스

프로세스 메트릭스는 소프트웨어의 개발 과정

이나 개발 환경의 속성을 평가하는 것으로, 자원의 메트릭스나 경험 레벨 메트릭스 등이 여기에 포함된다. 예를 들면, 자원의 메트릭스에는 프로그래밍 경험이나 개발 유지비용 등이 속하고, 경험 레벨 메트릭스의 예로서는 개발 팀이 사용하고 있는 프로그래밍 언어의 사용년수, 개발된 프로그램이 프로그래밍 팀에 소속되어 있는 년수나 유사한 소프트웨어를 개발한 경험년수 등이 속한다.

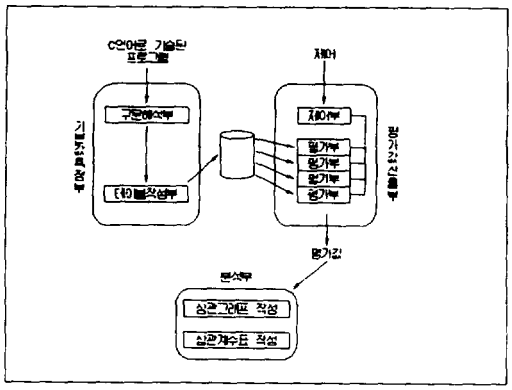
(2) 프로덕트 메트릭스

프로덕트 메트릭스는 제품으로서의 소프트웨어 자체의 특질을 평가하는 것으로, 소프트웨어 생산물의 사이즈(코드의 행수나 프로그램의 토큰의 수 등), 논리 구조 복잡성(흐름 제어, 중첩의 깊이, 재귀 등), 데이타 구조 복잡성(사용된 변수의 수 등), 이들의 조합 등이 포함된다.

4.2 프로덕트 메트릭스의 분류

(1) 사이즈 메트릭스

모든 프로그램은 사이즈라고 하는 공통의 특질을 가지고 있다. 따라서 사이즈 메트릭스는 행수라고 하는 메트릭스와 유사한 개념이다. 프로그램의 사이즈를 표현하는데는 가능한 많은 방법이 있다. 여기에서는 소프트웨어의 특질을 표현하는 역할을 하게된다. 프로그램의 사이즈를 나타내는 메트릭스의 예로는 코드의 행수, 토큰의 수, 함수의 수, 이런 기본적인 것을 조합시킨 것 등이 있다.



(그림 4) 시스템의 구성
(Fig. 4) Construction of system

(2) 논리구조 복잡성

프로그램의 논리구조는 다른 입력 데이타나 내부의 계산값에 의존하여 다른 동작을 하는 것을 허용한다. 테스트와 관련된 동작이 모여 프로그램을 구성하며, 그것을 알고리즘이라 한다. 알고리즘의 구조는 흐름도 또는 흐름그래프라고 부르는 유향 그래프로 표현된다.

5. 소프트웨어 복잡성 측정 시스템

5.1 구 성

본 소프트웨어 복잡성 측정 시스템은 C 언어로 작성된 원시 프로그램을 입력으로 하여 복잡성 모델에 포함된 여러 요인항목들의 값을 산출하기 위해 필요한 기본적인 값(metrics)들을 측정하는 기본값 측정부와 기본값 측정 결과로 구성된 기본값 테이블을 사용하여 각각의 요인항목의 값을 산출해내는 평가값 산출부로 이루어져 있다. 시스템의 구성을 (그림 4)에 나타내었다.

시스템을 구성하는 각 부분은 다음과 같은 기능을 한다.

5.1.1 기본값 측정부

복잡성 모델로 설정한 척도들의 값을 산출하기 위해 필요한 프로덕트 메트릭스를 산출하여 메트릭스 테이블을 작성하는 부분이다.

(1) 구문해석부

처리하고자 하는 소스 프로그램을 스캔하는 가장 작은 단위의 함수에서는 소스 프로그램으로부터 한 문자씩 읽어 들여 처리한다. 단일 문자에 대한 처리를 위해서 아스키 코드에 대한 문자 테이블이 존재한다. 읽어 들여 인식된 문자를 하나의 토큰을 이룰 때까지 문자열로 통합한다. 통합된 문자는 키워드 테이블 및 매크로 키워드 테이블의 내용과 비교하여 토큰이 키워드나 매크로 키워드인지를 검사한다. 테이블 내에 존재하지 않는 토큰일 경우에는 식별자로 인식된다. 스캔 과정에서 생성된 토큰은 키워드나 매크로 키워드일 수도 있고, 식별자나 함수명, 또는 상수일 수

도 있다. 식별자, 함수, 상수 등은 전역 또는 지역 속성을 가지며, 나타난 갯수나 횟수 등의 속성을 갖는다. 이러한 속성의 처리를 위해 시스템, 화일, 함수 레벨에 대해 각각 심벌 테이블을 갖는다. 심벌 테이블의 자료구조는 각 레벨이 모두 동일하다. 심벌 테이블은 변수, 함수, 상수, 라이브러리 함수 등에 대한 전역 또는 지역특성 및 나타난 횟수 및 갯수에 대한 정보를 유지한다.

(2) 테이블 작성부

구문해석부를 통해 측정된 메트릭스는 내부적으로 포인터를 이용한 복잡한 형태의 자료구조에 저장되며 외부적으로도 화일로 저장되어 복잡성 모델의 값을 측정할 때 사용된다.

5.1.2 평가값 산출부

기본값 테이블에 저장되어 있는 프로덕트 메트릭스의 정보를 이용해서, 복잡성 척도에 포함되어 있는 요인항목들의 평가값을 산출한다. 평가값의 산출은 각 척도에 대응하는 평가부에 의해 행해진다.

(1) 제어부

제어부는 복수의 평가부 중에서 어떤 평가부들을 선택하여 측정하며 다수의 요인항목을 갖는 모델인 경우는 어떤 요인항목들을 선택할 것인지를 결정하는 부분이다. 제어 화일은 다음과 같은 구성을 갖는다. (그림 5)에서 평가부는 측정하고자 하는 복잡성 모델을 나타내며 요인항목은 복잡성 모델에 속하는 요인항목 중 측정하고자 하는 요인항목을 선택하는 것이다.

평가부	요인항목
1	1 3
2	1
3	1

(그림 5) 제어 화일의 구성
(Fig. 5) Construction of control file

(2) 평가부

평가부는 기본값 측정부에서 측정한 프로덕트 메트릭스를 이용하여 복잡성 모델의 요인항목들의 값을 측정하는 부분이다. 프로덕트 메트릭스 테이블에 저장되어 있는 값을 사용하기 위해 메트릭스 값에 순차적인 인덱스가 부여되고 복잡성 모델의 요인항목에 필요한 값들은 인덱스를 이용하여 추출되어 측정에 사용된다. 측정되는 복잡성 모델은 Halstead의 모델과 McCabe의 모델, 그리고 D 척도이다. 하나의 화일을 이루는 함수들에 대해 복잡성 모델의 각 요인항목들을 적용하여 측정값을 구하고 함수들의 순서에 따라 측정값을 배열하여 화일로 만든다.

(3) 분석부

분석부는 복잡성 모델의 요인항목들에 대한 측정값들이 저장된 화일들을 이용하여 상관 그래프나 상관계수표를 출력함으로써 복잡성 모델간의 성능비교와 요인항목들간의 관계를 쉽게 파악할 수 있도록 하는 부분이다.

5.2 기 능

본 소프트웨어 복잡성 측정 시스템은 C 언어로 기술된 소스 화일군을 입력으로 한다. 하나의 시스템은 하나 이상의 화일로 이루어져 있다. 이러한 다수의 화일을 실행시에 함께 나열함으로써 한번에 평가가 이루어진다. 소스 화일을 구문분석하여 프로덕트 메트릭스를 측정하고 테이블로 만든다. 측정은 다음과 같이 3개의 레벨로 이루어진다.

(1) 시스템 레벨

시스템이 거대화 될 경우, 다수의 화일로 분할하여 하나의 시스템을 이루도록 만들 수 있다. 각각의 화일은 독립적으로 컴파일될 수 있으며 링크 과정에서 하나의 시스템으로 통합된다. 이러한 전체 시스템의 관점에서 측정된 메트릭스를 시스템 레벨의 메트릭스라고 한다.

(2) 화일 레벨

하나의 화일은 하나 또는 그 이상의 함수로 구성되어 있으며, 화일들이 모여서 하나의 시스템을 이룬다. 이러한 화일의 관점에서 측정된 메트

릭스를 화일 레벨의 메트릭스라고 한다.

(3) 함수 레벨

함수는 시스템을 구성하는 가장 작은 단위로서 다른 함수를 호출해서 사용하거나 다른 함수로부터 호출을 받아 사용될 수 있다. 이러한 함수의 관점에서 측정된 메트릭스를 함수 레벨의 메트릭스라고 한다.

5.3 외부사양

본 소프트웨어 복잡성 측정 시스템은 소스 프로그램을 분할하여 10개의 프로그램으로 구성되어 있다. 각 프로그램에서 서로 참조해야 할 외부 함수와 외부 변수에 대한 선언과 자료구조들은 2개의 헤더화일로 구성되어 있다. 하나의 헤더화일에는 구조체를 이용한 자료구조가 선언되어 있고, 또 하나의 헤더화일은 자료구조를 선언한 헤더화일을 포함하며 외부선언이 필요한 변수 및 함수들을 선언하고 있다. 각각의 프로그램은 기능별로 작업을 분담하며, 다수의 함수들로 구성되어 있다. 〈표 1〉에 본 소프트웨어 복잡성 측정 시스템에 대한 화일 구성을 나타내었다.

5.4 측정할 수 있는 메트릭스

구현된 복잡성 측정 시스템에서 측정할 수 있는 메트릭스를 <표 2>에 나타내었다. 각 메트릭스는 기본적으로 3개의 레벨로 측정 가능하지만 측정하여도 그 값에 의미가 없는 경우는 측정을 생략한다. 각 메트릭스가 어떤 레벨을 측정할 수 있는지를 표에 함께 나타내었다.

〈표 1〉 소프트웨어 복잡성 측정 시스템의 화일 구성
(Table 1) File construction of software complexity evaluation system

mtccount.c	메트릭스 계측 프로그램
mtcblock.c	외부선언해석 프로그램
mtcend.c	메트릭스 테이블 종료작업 프로그램
mtcexp.c	식해석 프로그램
mtcinit.c	메트릭스 테이블 초기화 작업 프로그램
mtclib.c	시스템 라이브러리 프로그램
mtcscan.c	문자열을 토큰으로 잘라내는 프로그램
mtcst.c	구문해석 프로그램
mtctab.c	식별자 테이블 프로그램
mtcput.c	계측된 메트릭스 테이블의 출력 및 복잡성 모델의 요인항목 값 산출
mtcdef.h	시스템 정의 헤더 화일
mtcstruct.h	구조체 정의 헤더 화일

〈표 2〉 메트릭스와 측정가능한 레벨의 일람표
(Table 2) Table of metrics and measurable level

메트릭스	level	메트릭스	level
행수(LOC)	1 2 3	사용자 정의 정수참조	1 2
문의 총수	1 2 3	함수	1 2 3
코멘트 행	1 2 3	함수호출	1 2 3
공백행	1 2 3	정의함수	1 2
실행가능문	1 2 3	정의함수호출	1 2 3
선언문	1 2 3	피호출함수	1 2 3
분기문	1 2 3	함수피호출	1 2 3
반복문	1 2 3	정의함수의 최대행수	1 2
연산문	1 2 3	정의함수의 평균행수	1 2
입출력문	1 2 3	정의함수의 행수의 분산	1 2
점프문	1 2 3	정의함수의 전역변수의 최대	1 2
레이블문	1 2 3	정의함수의 전역변수의 평균	1 2
토큰문	1 2 3	정의함수의 전역변수의 분산	1 2
키워드수	1 2 3	정의함수의 지역변수의 최대	1 2
식별자	1 2 3	정의함수의 지역변수의 평균	1 2
변수	1 2 3	정의함수의 지역변수의 분산	1 2
변수 참조	1 2 3	중첩 레벨의 최대	1 2 3
변수 정의	1 2 3	중첩 레벨의 평균	1 2 3
전역변수	1 2 3	중첩 레벨의 분산	1 2 3
전역변수 참조	1 2 3	화일의 수	1
전역변수 정의	1 2 3	화일의 최대행수	1
지역변수	1 2 3	화일의 평균행수	1
지역변수 참조	1 2 3	화일의 행수의 분산	1
지역변수 정의	1 2 3	화일의 정의함수의 최대	1
상수	1 2 3	화일의 정의함수의 평균	1
상수 참조	1 2 3	화일의 정의함수의 분산	1
사용자정의 정수	1 2		

1. System level
2. File level
3. Function level

5.5 시스템의 수행 결과

(1) 프로덕트 메트릭스의 출력화일

프로덕트 메트릭스는 시스템 레벨, 화일 레벨, 함수 레벨의 3가지 형태로 출력된다. 다음의 결과는 시스템 레벨 메트릭스를 보여주고 있다.

```
System name : testprog.c

<STATEMENT>
Lines of code                    684
Total number of statement        454
```

Number of comment line	50
Number of I/O statement	0
Number of labeled statement	84
Number of selection statement	67
Number of iteration statement	21
Number of jump statement	59
Number of expression statement	223
Number of executable statement	311
Number of declaration statement	22
Number of blank line	35
<TOKEN>	
Total number of token	2572
Number of keyword	302
Number of identifier	682
Number of operator	0
<VARIABLE>	
number of variable	4
number of reference	19
number of write	10

⋮ ⋮ ⋮

(이하 생략)

(2) 복잡성 모델의 측정결과

모듈내의 복잡성 척도 중에서 Halstead의 어휘 사이즈, Halstead의 프로그램 길이, Halstead의 프로그램 볼륨, 사이클로매틱수, D 척도를 각 모듈에 적용한 결과를 〈표 3〉에 나타내었다. 〈표 5〉는 측정결과를 정규화한 것이다. 정규화 방법은 요인항목들간의 상관관계가 높다는데 착안해서 각 모듈에 대한
측정값을 해당 요인항목의 총점으로 나눈 후 전체값들의 분포가 1이상이 될 수 있는 수를 곱하여 나타내었다. 이를 공식화하면 다음과 같다.

$$\text{정규화된 값 } N = \frac{\text{각 모듈에 대한 측정값}}{\text{해당 요인항목의 총점}} * k$$

(단, k는 전체값들의 분포가 1이상이 될 수 있는 수)

각각의 복잡성 메트릭스는 서로 다른 관점에서 복잡성을 측정하고 있으므로 다수의 복잡성 메트릭스를 하나의 시스템으로 통합함으로써 다양한 관점에서 복잡성을 측정할 수 있는 시스템을 구축할 수 있게 된다. 여기에서 다수의 복잡성 메트릭스를 하나의 시스템으로 통합하려면 각각의 메트릭스가 복잡성에 영향을 주는 정도에 있어서 균등한 역할을 해야 한다는 관점에서 복잡성 메트릭스나 요인항목간의 상관관계를 구하여 요인항목들간의 관련성을 파악한다. 요인항목들간의 상관관계가 높다면 정규화된 각각의 요인항목은 복잡성에 있어서 서로 다른 관점에서 균등한 영향을 미치게 된다. 〈표 4〉에 복잡성 모델에 포함된 요인항목들간의 상관관계를 나타내었다.

〈표 3〉 선택된 복잡성 모델과 요인항목에 대한 측정 결과
〈Table 3〉 Measurement result about complexity model and factor item

모 듈 명	size	leng	vol	cyc	D
abstr_decl	15	45	175.8	8	17
para_decl	19	70	297.3	13	11
declarator	16	74	296.0	11	16
initializer	7	10	28.0	4	2
struct_decl	6	14	36.1	6	10
struct_spec	5	8	18.5	3	1
enum_decl	9	13	41.2	4	2
enum_spec	5	8	18.5	3	1
typedf_decl	6	6	15.5	3	2
decl_spec	14	65	247.4	4	2
funct_def	17	43	175.7	9	8
extern_decl	15	30	117.2	6	5
incl_macro	10	22	73.0	4	2
uc_nscan	3	6	9.5	2	1
def_macro	13	58	214.6	13	6
if_macro	5	8	18.5	3	3
ifdef_macro	2	4	4.0	2	1
macro_decl	11	41	141.8	6	4
mainprogram	7	33	92.6	3	2
TOTAL	185	561	2128.4	107	96
AVERAGE	9.7	29.5	112.0	5.6	5.1

size : 어휘 사이즈
leng : 프로그램 길이
vol : 프로그램 볼륨
cyc : 사이클로매틱 수
D : D 척도

〈표 4〉 각 요인항목간의 상관관계표
〈Table 4〉 Correlation table between each factor item

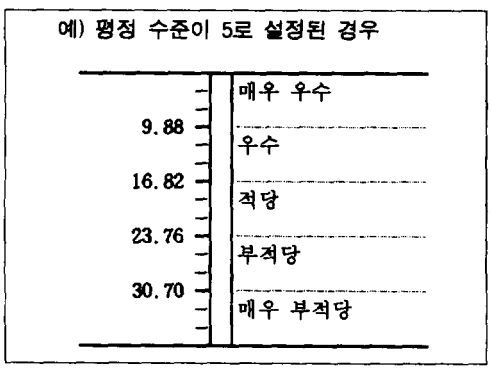
	size	leng	vol	cyc	D
size	1.0000	0.8778	0.9040	0.8167	0.6698
leng	0.8778	1.0000	0.9945	0.8110	0.6394
vol	0.9040	0.9945	1.0000	0.8332	0.6647
cyc	0.8167	0.8110	0.8332	1.0000	0.7488
D	0.6698	0.6394	0.6647	0.7488	1.0000

〈표 5〉 요인항목의 측정 결과에 대한 정규화
〈Table 5〉 Normalization about measurement result of factor item

모 듈 명	size	leng	vol	cyc	D	TOTAL	MEAN
abstr_decl	24	24	25	22	53	149	29.7
para_decl	31	37	42	36	34	181	36.2
declarator	26	40	42	31	50	188	37.6
initializer	11	5	4	11	6	38	7.6
struct_decl	10	7	5	17	31	70	14.1
struct_spec	8	4	3	8	3	27	5.3
enum_decl	15	7	6	11	6	45	9.0
enum_spec	8	4	3	8	3	27	5.3
typedef_decl	10	3	2	8	6	30	6.0
decl_spec	23	35	34	11	6	109	21.8
funct_def	28	23	25	25	25	126	25.1
extern_decl	24	16	17	17	16	89	17.9
incl_macro	16	12	10	11	6	56	11.2
uc_nscan	5	3	1	6	3	18	3.6
def_macro	21	31	30	36	19	138	27.5
if_macro	8	4	3	8	9	33	6.6
ifdef_macro	3	2	1	6	3	15	2.9
macro_decl	18	22	20	17	13	89	17.8
mainprogram	11	18	13	8	6	57	11.3

정규화된 결과를 바탕으로 평정 수준을 설정하기 위해 평균을 이용하였으며 19개의 모듈에 대한 복잡성 값을 5개의 구간으로 나누어 평정 수준을 (그림 6)과 같이 설정하였다.

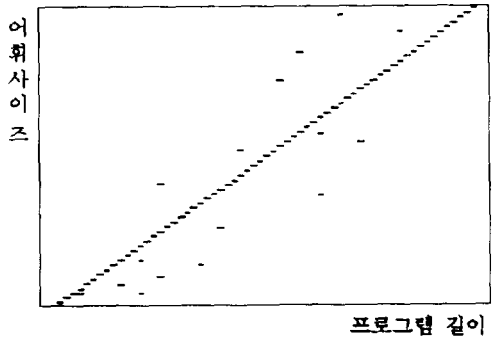
(그림 7)은 복잡성 모델의 측정 결과를 바탕으로 상관 그래프를 그린것이다. 상관 그래프를 통해 두 요인항목간의 상관관계를 직관적으로 파악할 수 있다.



(그림 6) 평정 수준의 설정
(Fig. 6) Setup of evaluation level

〈표 6〉 측정 결과에 대한 판정
〈Table 6〉 Decision of measurement result

모 듈 명	TOTAL	MEAN	판 정
abstr_decl	149	29.7	부 적 당
para_decl	181	36.2	매우부적당
declarator	188	37.6	매우부적당
initializer	38	7.6	매우 우수
struct_decl	70	14.1	우 수
struct_spec	27	5.3	매우 우수
enum_decl	45	9.0	매우 우수
enum_spec	27	5.3	매우 우수
typedef_decl	30	6.0	매우 우수
decl_spec	109	21.8	적 당
funct_def	126	25.1	부 적 당
extern_decl	89	17.9	적 당
incl_macro	56	11.2	우 수
uc_nscan	18	3.6	매우 우수
def_macro	138	27.5	부 적 당
if_macro	33	6.6	매우 우수
ifdef_macro	15	2.9	매우 우수
macro_decl	89	17.8	적 당
mainprogram	57	11.3	우 수



(그림 7) 상관 그래프의 예
(Fig. 7) Example correlation graph

본 소프트웨어 복잡성 측정 시스템은 소프트웨어의 복잡성을 측정하기 위해 필요한 메트릭스들을 측정하여 테이블화하고 메트릭스 테이블의 값을 이용하여 복잡성 모델의 요인항목들의 값을 산출한다. 각 요인항목들이 서로 다른 관점에서 복잡성을 측정하고 있으므로 요인항목들이 복잡성에 있어서 균등한 영향을 미치고 있는지를 파악하기 위해 요인항목들간의 상관관계를 구하고 그래프화하여 직관적으로 파악할 수 있도록 한다.

6. 결론 및 향후 연구과제

소프트웨어 유지보수 비용이 소프트웨어 라이프사이클에 소요되는 총 비용의 80%에 이르는 현 시점에서 유지보수 문제는 심각한 문제로 대두되고 있다. 이러한 관점에서 소프트웨어의 유지보수 비용을 줄일 수 있는 방안으로 소프트웨어의 품질평가를 통한 보수용이성, 즉 보수성 품질이 좋은 소프트웨어를 개발하는데 도움을 줄 수 있는 유지보수성 관리 방안이 널리 연구되고 있다[8]. 즉, 설계 및 개발단계에서부터 유지보수에 대한 문제를 고려함으로써 근본적으로 문제를 해결하고자 하는 것이다. 이미 소프트웨어의 품질과 관련된 메트릭스와 소프트웨어의 품질을 정량적으로 측정할 수 있는 소프트웨어 복잡성 측정 모델들이 다수 제안되어 있으며 품질평가를 위한 도구로 개발되어 활용되고 있다.

본 논문은 다수의 소프트웨어 복잡성 측정 모델들을 하나의 대상(소프트웨어)에 적용하여 평가함으로써 하나의 시스템 내에서 다수의 복잡성 모델이 소스 프로그램의 정적인 분석을 통해 측정하고자 하는 관점들을 함께 수용할 수 있는 시스템을 구현한 것이다. 하나의 모델내에 포함되어 있는 요인항목들은 서로 다른 관점에서 복잡성을 측정하고 있으므로 각 요인항목들이 통합되는 과정에서 각각이 평가에 있어서 차지하는 비중을 같은 비율로 하려면 모델간, 요인항목간의 상관관계를 측정하여 강한 상관관계를 나타내는 것들을 통합하여 복잡성의 값이 어떤 수준인가에 관계없이 각 모델이나 요인항목이 동일한 정도의 비율을 차지할 수 있도록 해야 한다. 이러한 관점에서 각 요인항목간의 상관관계를 측정하여 도표나 그래프를 이용하여 나타내었다.

본 논문에서 설계 및 구현한 소프트웨어 복잡성 측정 시스템은 소스 프로그램을 입력으로 받아들여 프로덕트 메트릭스를 측정한 후 테이블에 저장한다. 테이블에 저장된 프로덕트 메트릭스의 측정값은 복잡성 모델들의 요인항목 값을 산출할 때에 제어화일의 내용을 참조하여 필요한 프로덕트 메트릭스의 실값들이 추출되어 사용된다.

본 복잡성 측정 시스템은 측정 결과를 도표 형식으로 나타내고 있다. 현재 소프트웨어 개발의 추세는 인간이 좀 더 쉽게 이해하고 편리하게 사용할 수 있는 환경을 구축하는 방향으로 나아가고 있는만큼 출력 형태를 좀 더 다양하게 보여줌으로써 시각적으로 쉽게 결과를 이해할 수 있도록 구현 부분을 좀 더 보완해야 할 것이며, 소프트웨어에 대한 평가를 실시한 후 결과 제시에 그치지 않고 구체적인 개선 지침까지 제시할 수 있는 도구의 구현이 필요하다고 본다.

참 고 문 헌

[1] S. D. Conte, H. E. Dunsmore and V.Y. Shen, 'Software Engineering Metrics and Models', The Benjamin/Cummings Publishing, pp. 16-87, 1986.

[2] M. Shepperd, 'Software Engineering Metrics: Measures and Validations', McGRAW-HILL BOOK COMPANY, pp. 18-51, 1993.

[3] N. E. Fenton, 'Software Metrics-A rigorous approach', CHAPMAN & HALL, pp. 181-186, 1991.

[4] Roger S. Pressman, 'Software Engineering-A Practitioner's Approach', 3rd ed., Mc-Graw-Hill Inc., pp. 549-558, 1992.

[5] W. Harrison, K. Magel, R. Kluczny, and A. Dekock, "Applying Software Complexity Metrics to Program Maintenance", IEEE Computer, Vol. 15, No. 9, pp. 65-79, Sept. 1982.

[6] 양해술 외, '소프트웨어 품질보증과 품질평가 자동화 도구의 개발', '93장기기초연구과제 중간보고서, 한국통신연구개발단, 1993.

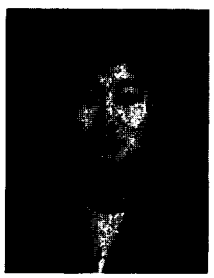
[7] 정호원, 양해술, 'ISO 9000 시리즈와 소프트웨어 품질 시스템(上)', 하이테크정보, pp. 170-1 75, 1993.

[8] 송영재, 'C언어로 구현한 소프트웨어 엔지니어링', 홍릉과학출판사, pp. 513-521, 1992.

[9] 양해술외, '품질관리방법론 및 지원도구의 개발', 과학기술처 STEP 2000과제, 중간보고서, 1995.

이 하 용

1993년 강원대학교 전자계산학과 졸업(이학사)
1995년 강원대학교 전자계산학과 소프트웨어공학 전공(이학석사)
1995년~현재 강원대학교 대학원 전자계산학과 박사과정
관심분야 : 소프트웨어공학(특히, S/W 품질보증과 품질평가, 객체지향 프로그래밍, 객체지향 분석과 설계, CASE).



이 용 근

1988년 강원대학교 자연과학대학 전자계산학과 졸업(이학사)
1994년 강원대학교 전자계산학과 소프트웨어공학 전공(이학석사)
1989년~92년 강원대학교 전자계산학과 조교
1995년~현재 강원대학교대학원 전산학과 박사과정
1994년~현재 한림전문대학 강사
관심분야 : 소프트웨어공학(특히, S/W 품질보증과 품질평가, 시스템통합 및 분산개발환경, 객체지향 분석과 설계).

박 정 호

1980년 성균관대학교 사범대학 졸업(학사)
1980년~82년 성균관대학교 경영대학원 정보처리학과(경영학석사)
1985년~87년 日本 오사까대학교 대학원 정보공학전공(공학석사)
1987년~90년 日本 오사까대학교 대학원 정보공학전공(공학박사)
1994년~현재 한국정보처리학회 학회지 편집위원장
1991년~현재 선문대학교 전자계산학과 조교수
관심분야 : 분산알고리즘, 소프트웨어공학, 시스템통합.

양 해 술

1975년 홍익대학교 공과 대학 전기공학과 졸업(학사)
1978년 성균관대학교 정보처리학과 정보처리 전공(석사)
1991년 日本 오사카대학교 기초공학부 정보공학과 소프트웨어공학 전공(공학박사)
1975년~79년 육군중앙경리단 전자계산실 시스템분석장교 근무
1984년~92년 성균관대학교 경영대학원 강사
1986년~87년 日本 오사카대학교 객원연구원
1993년~94년 한국정보과학회 학회지 편집부위원장
1980년~95.5 강원대학교 전자계산학과 교수
1994년~현재 한국산업표준원(IIS) 이사
1994년~현재 한국정보처리학회 논문지편집위원장
관심분야 : 소프트웨어 공학(특히, S/W 품질보증과 평가, SA/SD, OOA/OOD/OOP, CASE, SI), 소프트웨어 프로젝트관리.