프로그램의 효율성 측정 방법과 간소화

양 해 술†

요 약

소프트웨어에 대한 사용자의 요구를 충족시키기 위해 많은 기능을 가지는 방대한 규모의 소프트웨어가 개발되고 있다. 그러나 일반적으로 사용자는 소프트웨어의 부분적인 기능만을 사용하는 경우가 대부분이다. 많은 기능을 가지고 있는 소프트웨어에서 필요한 기능만을 남기고 불필요한 기능을 제거하여 구축할 경우 프로그램의 전체 사이즈가 줄어 실행 효율이 향상되고 그 프로그램을 유사 시스템에 재사용할 수 있으므로 결과적으로 소프트웨어의 품질을 높일 수 있게 된다. 소프트웨어의 품질에 관한 국제표준인 ISO/IEC 9126에서는 기능성, 신뢰성, 사용성, 효율성, 보수성, 이식성 등의 6항목을 정의하고 있다. 본 연구에서는 품질특성 중 효율성에 촛점을 맞추어 효율성의 측정을 위한 외부특성과 내부특성에 관련된 메트릭스를 제안하고 효율성을 향상시키기 위한 기본적인 방안으로 소스 코드에 대한 간소화 방안을 제안하였다. 또한, 제안된 효율성에 메트릭스에 대해 실제 개발 프로젝트에 적용하여 평가 결과를 기술하고 문제점과 개선방안을 예시하였다.

Efficiency Measurement Method and Simplification of Program

Hae-Sool Yang†

ABSTRACT

Softwares which have many functions to satisfy user's requirements is developing. But generally, users use partial functions of software. If we could construct software which leave useful functions and remove unuseful functions in software with many functions, we could enhance execution efficiency by reduction of program size and quality of software. There are 6 items in international standard ISO/IEC 9126 about quality of software. These are functionality, reliability, usability, efficiency, maintenance and portability. In this study, we proposed metrics for measurement of efficiency and simplification method for source code. And we described products evaluation result and indicated problem and progress method for practical development project about proposed efficiency metrics.

1. 서 론

소프트웨어 개발에 의한 업무 전산화 과정에서 소프트웨어에 대한 사용자의 인식이 높아지고 이에 비례하여 요구가 증대함에 따라 소프트웨어가 대규모화되면서 다양하고 복잡한 기능들을 갖추게 되었다[1, 2]. 이에 따라 소프트웨어의 개발비용이 비약적으로 높아지게 되었으며 특히 생명주기 단계 중에서 유지보수 단계가 차지하는 비율이 80%를 넘게 되어 소프트웨어의 품질에 대한 중요성이 부각되고 있다[3, 4, 9, 10].

현재 국내에서는 실용화하기에 적합한 수준의 품질관리 바업론의 정립이 미흡한 실정이며 이에 산학

† 종신회원 : 한국소프트웨어품질연구소(INSQ) 소장
논문접수 : 1997년 7월 8일, 심사완료 : 1997년 10월 18일

연을 중심으로 품질관리 방법론에 관한 연구가 활발히 진행되고 있다. 국제적으로도 품질특성에 관한 표준 ISO/IEC 9126이 정의되어 있으나 실용화하여 소프트웨어의 품질평가에 적용할 수 있는 세부단계의 수준까지 체계화되어 있지 못하며 현재 품질특성을 세분화하여 정량적으로 측정할 수 있는 실용적인 메트릭스에 대한 연구가 진행되고 있다[5, 8, 10, 15, 16]. 국제품질표준 ISO/IEC 9126은 품질특성으로 기능성, 신뢰성, 사용성, 효율성, 보수성, 이식성의 6항목을 정의하고 있다[13, 14, 18]. 그리고 개발되는 소프트웨어가 여러 기능을 갖추며 대규모화되는 경향인데 비해 사용자는 소프트웨어에 내장되어 있는 기능 중 극히 일부만을 이용하고 있는 것이 현실이다. 예를 들어, EMACS 에디터는 수백개의 기능을 가지고 있지만 보통 몇십개의 정해진 기능만을 이용하고 있다. 다기능 소프트웨어에서 사용자가 거의 이용하지 않는 기능들을 제거하고 불필요한 입출력을 최소화한 소프트웨어를 구축할 수 있다면 소프트웨어의 사이즈를 줄여 실행효율의 향상을 기대할 수 있다.

본 연구에서는 품질 주특성 중의 하나인 효율성에 초점을 맞추어 효율성을 정량적으로 측정할 수 있는 메트릭스를 제안하고 메트릭스에 정의한 바에 따라 효율성을 향상시키기 위한 방안을 기술하였다. 즉, 본 논문의 제2장에서는 관련 연구와 문제점에 대해 기술하였으며, 제3장에서는 효율성을 측정하기 위한 메트릭스와 기본적인 효율성 향상 방안으로 기존 프로그램의 기능을 입출력 값의 범위를 제한 함으로써 축소시킬 수 있는 방법을 제안하였다. 또한, 4장에서는 프로그램의 간소화 규칙 및 관련 기술과 프로그램의 간소화 사례로서 방법과 구조에 대해 기술하였다. 그리고, 5장에서는 제안 메트릭스의 타당성을 입증하기 위해 실제 개발 소프트웨어 프로젝트에 적용한 결과를 서술하였으며 끝으로, 본 연구의 결과와 향후 연구과제에 대해 기술하였다.

2. 관련 연구와 문제점

소프트웨어 품질에 관련된 의식의 변화로서 초기의 품질 무의식 시대에서 품질문제 표면화 시대와 체계적 품질관리 시대를 거쳐 자발적 품질관리 시대로 진전될 것이라는 기대와는 달리 소프트웨어 품질관리체계에 대해 지금까지 생명주기 단계별로 체계적인 지원 시스템이나 품질관리 방법론이 제대로 정립되어 있지 못한 것이 현실이다[15, 16]. 본 장에서는 소프트웨어 생명주기 단계와 관련된 품질 관련 연구결과들을 살펴보고 기존의 연구에서 나타나는 문제점들을 기술하였다.

2.1 소프트웨어의 품질

소프트웨어의 국제품질표준으로 ISO9001, Tickit과 같은 품질시스템이 전세계적으로 적용되고 있으며 이와 함께 프로세스 능력평가과 제품의 품질평가를 포함한 품질관리 방법론과 평가 메트릭스에 대한 관련 연구결과는 다음과 같다[8, 15, 16].

(1) 프로세스의 능력평가

소프트웨어를 개발하기 위한 소프트웨어 프로세스의 전체적인 능력을 정량적으로 평가하기 위해 필요한 평가 척도와 그 측정 평가방법으로 구성된 프로세스 메트릭스에 대해서는 명확하게는 정의되지 않고 있다. 즉, 지금까지 소프트웨어 프로세스의 능력은 소프트웨어 제품의 품질을 평가한 결과를 기초로 간접적으로 평가되어 왔다.

그러나, 1980년 후반부터 소프트웨어 프로세스가 가지는 총합적인 능력을 직접적으로 평가, 개선하도록 하는 경향이 강해지고 있으며 대표적인 소프트웨어 프로세스 평가모델로서는 다음의 4가지가 있다.

- CMM(Capability Maturity Model): 미국 카네기 대학 소프트웨어 생산기술연구소(SEI)에서 1987년에 제창된 프로세스 성숙도 모델로 현재도 SEI에서 개선하기 위해 연구개발을 행하는 동시에 보급 활동을 계속하고 있다.
- SPICE(Software Process Improvement and Capability dEtermination): 소프트웨어 프로세스의 개선과 능력의 결정을 목적으로 하는 프로세스 평가 모델로서 이 모델을 기초로 한 국제 표준화의 활동이 ISO/IEC JTC1 SC7/WG10에서 진행되고 있다.
- Bootstrap: 유럽의 ESPRIT 프로젝트에서 연구되고 있는 소프트웨어 프로세스 평가방식이다.
- ISO 9000-3: 하드웨어 및 시스템의 개발에 관련된 표준인 ISO 9001을 소프트웨어의 개발 프로

젝트에 적용하기 위한 가이드이다.

(2)제품의 품질평가

첫째, ISO/IEC 9126과 관련된 품질특성, 부특성, 내부특성의 구축 및 상호관계 확립

둘째, 생명주기 전단계에 걸친 품질평가 체계 및 평가 메트릭스의 개발

셋째, 구현단계의 소스코드에 대한 품질평가 모델로서 다음의 5가지 모델을 설정

- 기능성 모델:모듈 하나를 유지하는 기능의 수를 측정하여 기능 사이즈로 정량화한 것으로, 소스 프로그램의 각각의 모듈이 유지하고 있는 기능이 작으며 하나의 모듈당 하나의 기능을 가지고 있는 것이 바람직하다는 점을 고려한 모델이다.
- 이해용이성 모델:소스 프로그램의 읽기 쉬움을 정량화한 것으로 작성된 소스 프로그램이 시각적으로 읽기 쉽다면 프로그램을 빠르게 이해할 수 있음을 고려한 모델이다.
- 복잡성 모델:소스 프로그램의 논리의 복잡성을 정량화한 것으로 소스 프로그램의 논리가 단순 명쾌한 것이 바람직하다는 생각을 기초로 하는 모델이다.
- 구조화 모델:모듈간의 결합 복잡성을 정량화한 것으로 일반적인 소프트웨어가 다수의 모듈로 구성되어 있음을 착안하여, 모듈 사이의 관련성이 단순 명쾌한 결합 모델군으로 되어 있는 것이 바람직하다는 생각을 기초로 한다. 단순 명쾌한 결합이라면 보수할 때 영향을 주는 범위를 명확히 알 수 있다.
- 객체지향 모델:객체지향 프로그램에 대한 품질평가 모델로 객체지향 언어인 C^{++}로 작성된 소스 프로그램에 관해서 객체와 상속에 관련된 모듈 분석을 위한 메트릭스인 결합도와 응집도를 구한다.

넷째, C와 C^{++} 언어로 개발된 소스코드의 품질을 평가하는 자동화 도구의 개발에 관련된 연구 등이 있다.

2.2 소프트웨어 품질관리의 문제점

소프트웨어 생명주기 단계와 관련된 품질관리 분야에 대한 연구동향과 문제점들을 품질특성 모델, 메트릭스, 생명주기 공정, 상위공정의 품질평가에 중점을 두고 살펴보면 다음과 같다.

(1)품질특성 모델

지금까지 개발되어 있는 품질특성 모델은 프로그램 소스에 대한 복잡성이나 이해용이성, 모듈간의 관련성 등 대부분이 프로그램에 관련된 일부분의 특성을 평가하는 것이 대부분이고 소프트웨어 생명주기 전체에 관해 단계별로 체계적으로 정리된 모델이 적다.

(2)메트릭스

소프트웨어 품질특성 모델의 대부분이 구현과 테스트 공정을 고려한 모델이기 때문에 프로그램의 시험공수나 버그수에 관련된 것 소스레벨의 기능 사이즈, 구조, 이해 용이성 및 복잡성 등의 메트릭스 설정에 주안점을 두고 있다[15].

(3)초기공정의 품질

초기공정의 산출물인 명세서의 품질에 대해서는 DR(Design Review) 등으로 점검하는 경우가 대부분이고 체계적이며 정량적인 품질평가 방법을 적용하고 있지 않다.

(4)각 공정간의 연결

대부분의 품질특성 모델과 메트릭스는 생명주기 각 단계별 단일공정에서 사용하도록 되어있으며 특정 공정의 산출물이 다음 공정의 산출물에 미치는 영향에 대한 검토가 충분하지 못하다.

3. 효율성 메트릭스의 측정과 향상 방법

프로그램의 효율성을 측정하기 위한 메트릭스의 개발과 관련 ISO/IEC 9126의 품질특성 6항목 중 하나인 효율성을 바탕으로 이를 세분화한 품질부특성과 관련 내부특성을 체계화하였다.

품질특성은 크게 외부특성과 내부특성으로 분류할 수 있으며, 외부특성은 사용자 관점 또는 제품평가를 의식한 특성이고, 내부특성은 개발자 관점 또는 개발 프로세스의 평가를 의식한 특성이다. 즉, 외부특성이란 소프트웨어의 내부(예를 들면 구조나 명령어 등)를 살펴볼 수 없고, 소프트웨어를 이용하기 위한 이해 정도를 나타내는 소프트웨어 특성이며, 내부특성이란 복잡한 형태의 소프트웨어 내부를 살펴봄으로써 초기에 이해할 수 있는 소프트웨어 특성이다. 외부특성은 소프트웨어의 실제 운용환경에서 품질평가를 위한 척도이다. 따라서 외부특성은 다음과 같은 상황에서 적용가능한 척도이다.

①인수 시험에서의 품질평가로 조달자가 주체가 된다.

②다수의 대안이 있는 소프트웨어의 구매에서 대안들에 대해 품질평가하는 것으로 구매자가 주체가 된다.

③개발 후반부인 시험과 운영 단계에서 실시하는 품질평가로 개발자가 주체가 된다.

④제3자 품질인증을 위한 시험연구소에서 실시하는 품질평가로 시험소가 주체가 된다.

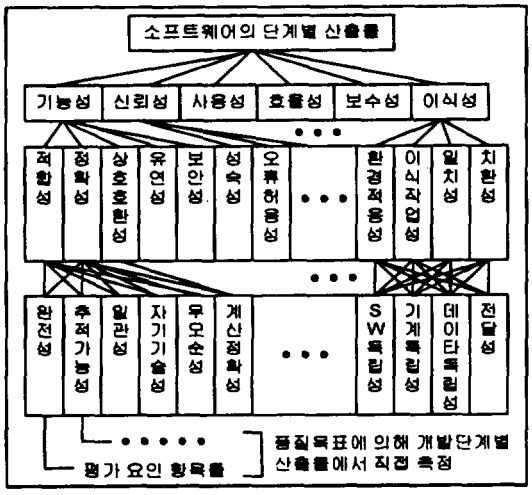
내부특성은 소프트웨어가 실제로 작동하기 이전 단계에서 품질을 측정하고 평가하기 위한 척도를 정의하는 것으로 다음과 같은 상황에서 적용 가능한 척도이다.

①소프트웨어 개발자가 주체가 되는 품질목표의 기획 및 선정

②내외부 평가자가 주체가 되는 시험이나 검토에서의 소프트웨어 품질평가

③생명주기 전단계에 걸쳐 개발되는 각 단계별 산출물에 대한 품질 측정 및 평가

현재, 본 연구와 관련해서 ISO/IEC 9126에 정의된 6항목의 품질특성을 세분하여 품질부특성 21항목을 정의하였고 또한, 관련 내부특성 40항목을 설정하여 품질을 평가할 수 있는 측정표를 구축하였다. (그림 1)은 품질특성, 품질부특성, 내부특성간의 관계를 나타내고 있다. 이러한 관계에 따라 품질부특성의 메트릭스로부터 직접 품질특성을 평가하거나, 내부특성을 평가한 후 (그림 1)의 관계에 따라 상향식으로 품질특성을 측정한다.

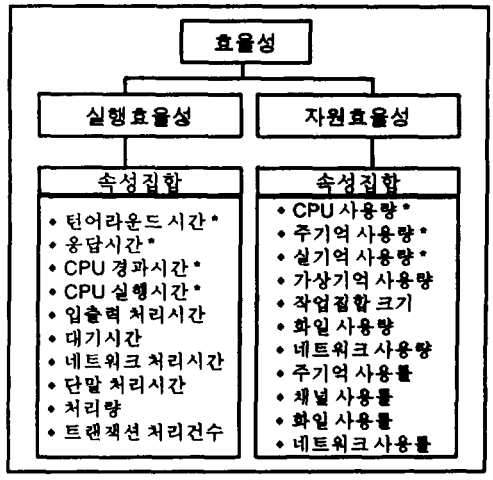


(그림 1) 품질특성, 부특성, 내부특성의 관계

(Fig. 1) Relation of Quality Characteristics, Subcharacteristics, Internal Characteristics

3.1 외부특성에 관한 효율성의 메트릭스

ISO/IEC 9126에는 외부특성에 관한 품질특성으로서 기능성, 신뢰성, 사용성, 효율성, 보수성, 이식성을 정의하고 있으며 이들을 좀더 세분하여 품질부특성을 정의하였다. 이중에서 본 연구의 주제인 효율성은 (그림 2)와 같이 명시적인 조건하에서 소프트웨어의 실행 레벨과 사용되는 자원 양자간의 관계를 나타내는 소프트웨어 속성의 집합을 의미한다[14, 15, 16]. 즉, 실행효율성은 턴어라운드 시간을 포함한 10가지의 속성집합으로 구성되며 자원효율성은 CPU 사용량을 포함한 11가지의 속성집합으로 매트릭스를 구축하였다.



(주) * : ISO 9126-2와 동일

(그림 2) 효율성 메트릭스의 구조

(Fig. 2) Structure of Efficiency Metrics

2.1.1 실행효율성의 메트릭스

실행효율성은 소프트웨어의 기능을 실행할 때 응답시간과 처리시간 그리고 기능수행에서 처리율과 관련된 소프트웨어의 속성으로 〈표 1〉과 같이 메트릭

스를 정의한다[11, 16].

〈표 1〉 실행효율성의 메트릭스
〈Table 1〉 Metrics of Execution Efficiency

속성집합	정의	측정식
1.턴 어라운드 시간(Tt) *	처리요구가 발생된 시점에서 처리결과가 얻어지기까지의 경과 시간	• Tt= $P_et - P_rt$ P_et : 처리결과 산출시간 P_rt : 처리요구 발생시간
2.응답시간 (Rt) *	컴퓨터 시스템에 대하여 조회 또는 요구 종료에서 응답초기까지의 경과시간	• Rt = $R_ft - R_et$ R_et:컴퓨터 시스템에서 응답 개시시간 R_ft:컴퓨터 시스템으로의 요구 종료시간
3.CPU경과 시간(Ct) *	프로그램의 시행개시에서 종료까지 필요한 경과시간	• Ct = $R_et - R_st$ R_et : 프로그램 실행종료시간 R_st : 프로그램 실행개시시간
4.CPU실행 시간(CRt) *	프로그램의 실행개시부터 종료까지 요구되는 CPU 시간	• CRt = $\sum_{i=1}^{n} P_i + \sum_{i=1}^{n} O_i$ Pi : 프로그램 실행에 필요한 CPU시간 Oi : 프로그램 실행에 따르는 OS나 관련 프로그램의 실행에 필요한 CPU시간
5.입출력 처리시간 (IOpt)	외부기억과의 입출력에 요구되는 시간	• IOpt = $\sum_{i=1}^{n}(IO1_i + IO2_i)$ $IO1_i$: OS에 의해서 입출력 처리명령이 발행된 시간 $IO2_i$: OS가 입출력 처리의 완료를 받는 시간
6.대기시간 (Wt)	프로그램의 실행이 OS나 자원에 의한 인터럽트에 의해서 대기한 시간	• Wt = 프로그램실행의 총 대기 시간
7.네트워크 처리시간 (NPt)	컴퓨터 센터와 단말간에서 전송의 개시 처리에서 종료처리까지 걸린 시간	• NPt = $\sum_{i=1}^{n}(CR_i + CS_i)$ CR_i : 센터 또는 단말에서 송신처리가 수신된 시간 CS_i : 센터에서 단말까지 또는 단말에서 센터까지 송신처리를 개시한 시간
8.단말처리 시간(TPt)	단말에서 처리의 개시에서 종료까지 요구된 시간	• TPt = $\sum_{i=1}^{n} P_i$ P_i : 단말처리시간
9.처리량(Pa)	단위시간내에 컴퓨터 시스템에 의해 수행된 업무량	• Pa = 컴퓨터 시스템에서 처리된 업무량/ 소요시간
10.트랜잭션 처리건수 (TPn)	단위 시간당 처리한 트랜잭션의 건수	• TPn = 트랜잭션 처리건수/ 소요시간

(주) * : ISO 9126-2와 동일

2.1.2 자원효율성의 메트릭스

자원효율성은 소프트웨어의 기능을 실행할 때 사용한 자원의 양과 그 사용시간을 나타내는 속성으로 〈표 2〉와 같은 메트릭스들을 정의하였다[11, 16].

〈표 2〉 자원효율성의 메트릭스
〈Table 2〉 Metrics of Resource Efficiency

속성집합	정의	측정식
1.CPU 사용량 (PUa) *	프로그램의 개시에서 종료까지 요구된 CPU 시간	• CPUa = $\sum_{i=1}^{n} P_i + \sum_{i=1}^{n} O_i$ Pi : 프로그램 실행에 필요한 CPU 시간 Oi : 프로그램실행에 따라 OS나 관련 프로그램의 실행에 필요한 CPU 시간
2.주기억 사용량 (MAa) *	프로그램의 실행에 필요한 프로그램 및 데이터 영역의 주기억량	• MAa = $CP_pa + CP_da$ CP_pa : 프로그램의 실행에 필요한 프로그램 영역만큼의 주기억량 CP_da : 프로그램의 실행에 필요한 데이타 영역만큼의 주기억량
3.실기억 사용량 (RMa) *	프로그램의 실행에 필요한 프로그램 및 데이타 영역의 실기억량	• RMa = $RMp - RMd$ RMp : 프로그램의 실행에 필요한 프로그램 영역만큼의 실기억량 RMd : 프로그램의 실행에 필요한 데이타 영역만큼의 실기억량
4.가상기억 사용량 (VMa)	프로그램의 실행에 필요한 프로그램 및 데이타 영역의 가상기억량	• VMa = $VMp + VMd$ VMp : 프로그램의 실행에 필요한 프로그램 영역만큼의 가상기억량 VMd : 프로그램의 실행에 필요한 데이타 영역만큼의 가상기억량
5.작업집합 크기(WSv)	프로그램 실행에 필요한 작업 집합 크기	• WSv = 프로그램의 실행에서 종료까지 필요한 크기
6.화일 사용량 (FUa)	프로그램이 사용하는 화일의 량으로 다음과 같이 측정한다.	• FUa = 프로그램이 사용하는 화일의 양
7.네트워크 사용량(Na)	송수신한 데이타 량	• Na = 송수신한 데이타 양
8.주기억 사용율(Mr)	일정 기간내의 주기억 사용 비율	• Mr = 주기억 사용량/ 모든 주기억량
9.채널 사용율(Cr)	일정 시간내에서 채널을 사용한 시간이 차지하는 비율	• Cr = 채널 사용시간/ 채널 사용 가능시간
10.화일 사용율(Fr)	일정 시간내에서 화일을 사용한 비율	• Fr = 화일사용할당량/ 모든 화일용량
11.네트워크 사용율(Nr)	일정 시간내에서 네트워크를 사용한 비율	• Nr = 실제로 송수신한 데이타 량/송수신 가능한 데이타 량

(주) * : ISO 9126-2와 동일

3.2 내부특성에 관한 효율성의 메트릭스

본 절에서는 40개의 내부특성 중 효율성에 관한 내부특성인 실행효율성과 자원효율성을 측정할 수 있는 측정표와 메트릭스에 대해 기술하였다. 실행효율성은 소프트웨어의 기능을 실행할 때의 응답시간, 처리시간, 처리량(throughput)의 효율을 양호하게 하는 성질이며 자원효율성은 소프트웨어의 기능을 실행할 때 사용하는 자원량과 사용시간을 절약하는 성질이다. 본 연구에서는 이와 같은 실행효율성과 자원효율성 메트릭스를 측정표로 구성하여 측정할 수 있도록 하였다. 즉, 각 내부특성을 요인항목, 세부항목, 요소

데이타로 분류하였으며 요인항목과 세부항목에 대해 가중치를 적용하여 효율성에 대한 품질을 평가할 때 사용자의 요구품질을 적용할 수 있도록 하였다[15, 16]. 따라서 소프트웨어 개발 산출물로부터 직접 추출할 수 있는 요소데이타의 값을 먼저 측정하고 세부항목과 요인항목의 값은 요소데이타의 값을 바탕으로 가중치를 적용하여 차례로 산출한다.

3.2.1 실행효율성의 측정표

실행효율성을 측정하기 위한 측정표는 〈표 3〉과 같으며 각 세부항목은 해당 요소데이타로 이루어진 계산식에 의해 평가된다. 요인항목과 세부항목은 중요도에 따라 가중치를 가지며 관련 항목들의 가중치 합은 1이 된다. 가중치의 선정에 있어서는 우선 사용자의 의견을 최우선으로 고려하고 부가적으로 시스템의 용도 및 가중치 관련 항목이 그 상위 항목에 끼치는 영향 정도 등을 고려한다. 또한, 품질평가 결과가 축적됨에 따라 일반적으로 취약한 결과를 보이는 항목에 대해 가중치를 좀더 높임으로써 향후 문제점의 수정 보완을 통한 품질향상을 아울러 고려하게 된다. 따라서 가중치는 고정적인 값이 아닌 평가 대상 및 사용자의 요구 수준에 따라 변화될 수 있는 값이다. 세부항목과 해당 가중치를 곱한 결과를 합산한 것이 요인항목에 대한 평가 결과가 되며, 요인항목의 값과 해당 가중치를 곱한 결과를 합산한 것이 내부특성인 실행효율성의 평가 결과가 된다.

〈표 3〉 실행효율성의 측정표
〈Table 3〉 Measurement Table of Execution Efficiency

실행 효율성			
요인항목	세 부 항 목	평가(예)	비 고
1.입출력 횟수 (0.4)	PDi1:각 기능들에서 처리를 위해 필요한 외부장치와의 입출력 횟수가 어느정도인가? (0.6)	79.5	요소데이타와 계산식으로 도출 (표 4 참조)
	PDi2:각 기능에서 처리를 위해 필요한 외부 데이타의 입출력 횟수는 어느 정도인가? (0.4)	83.2	상 동
	계 : 0.6*79.5 + 0.4*83.2 = 80.98		
2.동적 스텝수 (0.3)	PDi3:CPU로 처리되는 동적 스텝수(계산식)는 어느정도인가? (0.6)	75.4	상 동
	PDi4:CPU로 처리되는 동적 스텝중 호출하는 외부의 기능은 어느정도인가? (0.4)	73.7	상 동
	계 : 0.6*75.4 + 0.4*73.7 = 74.72		
3.평균 기동 시간 (0.3)	PDi5:각 기능별 평균 기동 횟수는 어느정도인가? (1.0)	82.6	상 동
	계 : 1.0*82.6 = 82.6		
총 계 : 0.4*80.98 + 0.3*74.72 + 0.3*82.6 = 85.40			

(주) 괄호 안의 수는 각 항목별 가중치

개발 산출물로부터 직접 평가할 수 있는 항목을 요소데이타라고 하며 각 세부항목의 값은 요소데이타로 구성된 수식을 이용하여 산출된다. 실행효율성의 각 세부항목에 관한 요소데이타와 계산식은 〈표 4〉와 같으며 계산식에 의해 세부항목은 0과 1사이의 값을 가지고 평가 결과를 백분율로 변환하여 세부항목의 평가 결과가 계산된다. 경우에 따라 세부항목의 값을 구하기 위해 필요한 요소데이타가 개발산출물에서 누락되어 추출할 수 없는 경우도 있다. 이러한 경우에는 평가 대상 프로젝트에서 이 요소데이타가 가지는 의미를 파악하여 불필요한 것이라면 측정 대상에서 제외하고 나머지 항목으로 가중치를 조정하여 평가하고, 누락된 것이라면 0점으로 평가한다.

〈표 4〉 실행효율성의 요소데이타와 계산식
〈Table 4〉 Element Data and Expression of Execution Efficiency

세부항목	요소데이타를 이용한 계산식
PDi1	각 기능에서의 외부장치와의 입·출력 처리의 횟수 / 각 기능에 필요한 모든 처리의 수
PDi2	처리를 위해 필요한 외부데이타 수 / 처리를 위해 필요한 모든 데이타 수
PDi3	CPU로 처리되는 동적 스텝수 / 소프트웨어의 모든 스텝수
PDi4	호출하는 외부 기능의 수 / CPU로 처리되는 모든 동적 스텝수
PDi5	기능별 기동횟수들의 총합 / 기능의 총수

3.2.2 자원효율성의 측정표

자원효율성을 측정하기 위한 측정표는 〈표 5〉와 같으며 각 세부항목은 해당 요소데이타로 이루어진 계산식에 의해 평가된다. 요인항목과 세부항목은 중요도에 따라 가중치를 가지며 관련 항목들의 가중치 합은 1이 된다. 세부항목과 해당 가중치를 곱한 결과를 합산한 것이 요인항목에 대한 평가 결과가 되며, 요인항목의 값과 해당 가중치를 곱한 결과를 합산한 것이 내부특성인 자원효율성의 평가 결과가 된다.

〈표 5〉 자원효율성의 측정표
〈Table 5〉 Measurement Table of Resource Efficiency

자원효율성			
요인항목	세 부 항 목	평가(예)	비 고
1.데이타 중복률 (0.7)	RDi1:데이타의 중복율 최소화하기 위한 처리 기능이 어느정도 제공되는가? (0.4)	72.5	요소데이타와 계산식으로 도출 (표 6 참조)
	RDi:데이타 관리에 대한 처리 기능이 어느정도 제공되는가? (0.3)	84.2	상 동
	RDi3:모듈별 각 데이타에 대한 이용 비율은 어느정도인가? (0.3)	79.4	상 동
	계 : 0.4*72.5 + 0.3*84.2 + 0.3*79.4 = 78.08		
2.출력 페이지 밀도 (0.3)	RDi4:출력 페이지당 정보량은 어느정도인가? (1.0)	75.9	상 동
	계 : 1.0*75.9 = 75.9		
총 계 : 0.7*78.08 + 0.3*75.9 = 77.43			

(주) 괄호 안의 수는 각 항목별 가중치

〈표 6〉 자원효율성의 요소데이타와 계산식
〈Table 6〉 Element Data and Expression of Resource Efficiency

세부항목	요소데이타를 이용한 계산식
RDi1	중복에 대한 처리가 제공되는 건수 / 데이타의 중복이 발생가능한 건수
RDi2	처리기능이 제공되는 데이타의 수 / 관리되어야 할 데이타의 종류수
RDi3	각 모듈별 데이타에 대한 액세스 수 / 모듈별 데이타에 대한 총 액세스 수
RDi4	의미있는 값을 갖는 자료의 문자수 / 출력된 페이지의 전체 문자의 수

개발 산출물로부터 직접 평가할 수 있는 항목을 요소데이타라고 하며 각 세부항목의 값은 요소데이타로 구성된 수식을 이용하여 산출된다. 〈표 6〉은 자원효율성의 각 세부항목에 대한 요소데이타와 계산식이며 계산식에 의해 세부항목은 0과 1사이의 값을 가지게 되며 평가 결과를 백분율로 변환하여 세부항목의 평가 결과로 한다.

3.3 효율성 향상 방법

프로그램의 효율성을 향상시키기 위한 기본적인 방법으로 프로그램의 다양한 기능들 중 사용되지 않는 기능들을 찾아 제거하거나 프로그램의 효율성에 관련된 품질특성 메트릭스에 따라 향상 방안을 모색하는 방법이 있다. 본 절에서는 프로그램 간소화에 관련해서 효율성 향상 방안을 기술하고 기타 메트릭스에 관련된 향상 방안을 살펴보았다. 입출력값을 제한하여 프로그램의 기능을 축소시키는 것이 프로그램 간소화이다. 모든 프로그램은 특정한 입력만을 처리하는 경우, 입력 중 일부의 값만 처리하는 경우, 일부분의 출력만 처리하는 경우 등의 목적에 맞춰 프로그램을 간소화 할 수 있다. 프로그램을 간소화하면 사이즈가 작아져 실행 효율이 좋아지고 다른 업무에도 재이용할 수 있는 이점이 있으므로 실행효율성과 자원효율성의 속성에 좋은 지표들을 제공할 수 있다. 프로그램의 간소화 방법은 다음과 같이 입력을 제한하여 간소화하는 경우와 출력을 제한하여 간소화하는 경우를 생각할 수 있다.

3.3.1 입력 제한

프로그램의 입력을 제한하면 프로그램내 분기문에서 분기방향이 일정하거나 반복문에서 반복 회수가 정해진다. 즉, 프로그램내 불필요한 문장이나 조건 판정 등의 논리를 생략하면 문장이나 변수가 필요없게 되어 프로그램을 간소화할 수 있다. 입력 제한은 다음과 같은 경우를 고려할 수 있다. 입력 데이타 값의

범위 제한과 고정 특정 값 배제 등의 입력을 제한하여 프로그램을 간소화하기 위해서는 처음 제한된 입력이 변수에 어떤 영향을 미치는가를 조사해야 한다. 즉, 프로그램의 제어 흐름과 데이타 흐름에 대한 조사가 필요하다. 입력 제한에 의해 값의 범위가 변화하고 특정 값에 고정된 변수가 있으면 당연히 그 변수를 참조하는 식과 계산 결과에 영향을 주게 된다. 그리고 프로그램 문장이나 조건 판정을 삭제하면 프로그램의 제어 흐름과 데이타 흐름에 영향을 주므로 당연히 변수나 그 변수에 관련있는 문장은 불필요하게 된다. 입력 제한에 의한 프로그램 간소화 절차는 다음과 같은 단계로 진행한다.

(단계 1) 제한시킬 입력을 정한다.
(단계 2) 범위가 변화되거나 값이 정해진 변수를 조사한다.
(단계 3) 그 변수를 참조하는 문장을 분석한다.
(단계 4) 수정 규칙을 적용하여 문을 수정한다.
(단계 5) 수정에 의해 새로운 범위를 변화시키는 변수나 불필요하게 된 변수문이 없는지 조사한다. 만약 있으면 (단계 3)으로 되돌아 간다.

3.3.2 출력 제한

출력에는 반드시 대응하는 출력문이 있다. 따라서 어떤 출력이 불필요하면 그에 해당되는 출력문은 불필요하게 된다. 즉, 어떤 출력을 제한하려면 그 출력에 영향을 주는 문장에 제한을 가하면 된다. 출력 제한은 다음과 같이 고려할 수 있다.

① 복수개의 출력값 중 일부만을 필요로 하는 경우
② 출력값을 어느 범위로 억제하고자 하는 경우

본 연구에서는 특정 출력값을 출력시키지 않는 전제의 출력제한에 대해서 고려한다. 출력되지 않는 값은 당연히 계산할 필요가 없다. 따라서 그 값을 계산하기 위해 기술되어 있는 문은 불필요하므로 삭제시켜 프로그램을 간소화할 수 있다. 불필요한 문을 삭제하면 이와 관련된 문장이나 변수가 불필요하게 된다. 이와 같이 출력을 제한하여 프로그램을 간소화하기 위해서는 출력에 대응하는 출력문을 조사한다. 출력문에 영향을 주는 문장, 즉 출력값 계산에 필요한 문장을 모두 추출하면 프로그램을 간소화할 수 있다. 그리고 프로그램 슬라이스는 프로그램내에서 어떤 문장 실행에 영향을 주는 모든 문을 추출하는 기술이다. 즉, 출력문에서 프로그램 슬라이스를 구하는 것은 출력을 제한하여 프로그램을 간소화하는 것과 같으며 출력 제한에 의한 프로그램 간소화 절차는 다음과 같다.

(단계 1) 필요한 출력을 모두 정한다.
(단계 2) 필요한 출력에 대응하는 출력문을 모두 조사한다.
(단계 3) 얻어진 출력문에서 프로그램 슬라이스를 구한다.
(단계 4) 어떤 출력문 슬라이스에도 포함되지 않은 문을 조사한다.
(단계 5) 얻어진 문을 삭제한다.
(단계 6) 불필요한 문의 삭제에 대해 수정 규칙이 적용 가능한 문장을 수정한다.

---

4. 프로그램의 간소화 규칙과 방법

본 장에서는 프로그램을 간소화하는데 적용되는 정적 분석 기법과 간소화시 필요한 수정 규칙, 불필요한 변수의 판단 기준에 대해 기술한다.

4.1 프로그램의 부분 계산

프로그램에 의해 입력된 데이타를 모두 처리하는 것을 전체계산이라고 하며, 일부 입력 데이타가 주어졌을 때 그 데이타만을 이용하여 계산을 행하는 것을 부분계산이라고 한다[1, 2, 6]. 즉, 주어진 입력을 토대로 실행 가능한 것을 행하고 실행할 수 없는 것은 그대로 남겨 둔다. 보통 부분 계산을 행하는 프로그램은 전체계산 프로그램보다 효율이 좋은 간소화된 프로그램이다. 부분계산하여 얻어진 결과에 나머지 입력 데이타를 주어 얻어진 결과는 전체 계산 프로그램을 실행하여 얻어진 계산 결과와 같다. 예를 들어 (그림 3)의 (a)의 프로그램에 y=1을 준 경우를 생각하면, 1행의 if문 조건식(x > 0)을 계산해도 값은 정해지지 않는다. 2행의 if문 조건식(y > 0)을 계산하면 참(true)이 된다. 따라서 else절(3행)은 실행되지 않아 불필요하게 된다. x의 대입문 우변(x + y)은 y값을 알 수

있으므로 x +1이 된다. 5행의 대입문 우변(-x)은 변하지 않는다. 따라서 부분 계산 프로그램은 (그림 3)의 (b)와 같이 된다. 일반적으로 부분계산에서는 반복문을 몇회 전개하는지 등의 계산 전략을 정할 필요가 있다. 여기에서는 다음 절에서 기술한 바와 같이 식 계산, if문 등의 단순한 수정을 행하였다.

```
(a) 프로그램 (y=1)
if x > 0 then
if y > 0 then x = x + y;
       else x = x - y;
else
   x = -x;

(b) 부분계산 프로그램 (y=1)
if x > 0 then
       x = x + 1;
else
   x = -x;
```

(그림 3) 프로그램 부분계산의 예
(Fig. 3) Example of Program Part Evaluation

4.2 프로그램 간소화시의 수정 규칙

프로그램을 부분적으로 평가하면 분기문, 반복문 등에서 실행되는 경로가 같은 경우가 발생한다. 이와 같은 경우, 실행되지 않는 경로는 삭제할 수 있으므로 대응하는 분기문, 반복문을 수정하는 것이 가능하다. 또, 수정을 하면 새로운 수정이 발생하는 경우도 생긴다. 예를 들어, (그림 4)의 프로그램에 y=1을 부여한 경우, 1행의 if문 조건식(x > 0)을 계산해도 값은 정해지지 않는다. 2행의 if문의 조건식(y=0)을 계산하면 거짓(false)이 된다. 따라서 then 절은 실행되지 않는다. 또한 else절도 없으므로 2행의 if문은 필요하지 않게 되어 삭제할 수 있다. 2행의 if문을 삭제하면 1행의 if문 then절은 거짓문이 된다. 따라서 2행의 if문도 필요하지 않게 되어 삭제할 수 있으므로 얻어진 프로그램은 거짓이 된다. (그림 5)에 본 방법에 이용한 수정 규칙의 예를 기술하였다.

```
1. if x > 0 then
2.     if y > 0 then x = x * x;
```

(그림 4) 새로운 수정의 발생 예
(Fig. 4) Example of New Modification Outbreak

```
1. if true then Statement --> Statement
2. if false then Statement --> 0
3. if true then Statement-A else Statement-B
                              --> Statement-A
4. if false then Statement-A else statement-B
                              --> Statement-B
5. while false do Statement --> 0
6. if exp then 0 --> exp
7. if exp then Statement else 0 -->
                        if exp then Statement
8. if exp then 0 else Statement -->
                        if not exp then Statement
9. if exp then 0 else 0 --> exp
```

(그림 5) 수정 규칙의 예
(Fig. 5) Example of Correction Rule

4.3 프로그램 슬라이스

프로그램 슬라이싱(pogram slicing)은 프로그램내 문장들간의 의존관계를 조사하여 문 실행에 영향을 주는 모든 문장을 추출하는 기술이다. 추출된 문의 집합을 슬라이스(slice)라 한다. 슬라이스에는 프로그램을 정적으로 해석하여 얻어진 정적 슬라이스와 동적으로 해석하여 얻어진 동적 슬라이스가 있다. 본 연구에서는 정적 슬라이스를 프로그램 간소화에 이용하였다. 어떤 문장에 대한 정적 슬라이스란 다음에 기술한 두 가지의 의존 관계에 의해 변수에 도달하는 모든 문의 집합을 의미한다.

- 데이타 의존:문 s1에서 어떤 변수 v의 정의가 v를 사용하고 있는 문 s2에 도달할 때, 문 s1과 문 s2는 데이타 의존(data dependence)의 관계이다.
- 제어 의존:문 s1이 분기문이거나 반복문이고, 문 s2의 실행 유무가 문 s1의 실행 결과에 직접 의존할 때, 문 s1과 문 s2는 제어 의존(control dependence) 관계이다.

위와 같은 의존관계를 이용하여 프로그램을 그래프화 한 것을 프로그램 의존 그래프(program dependence graph)라 한다. 어떤 문에 대한 정적 슬라이스는 그 문의 실행 결과를 보존하는 완전한 프로그램을 구성하는 특징을 가진다.

4.4 불필요한 변수의 검출 방법

프로그램이 수정되면 본래의 프로그램에 존재한 변수의 정의나 참조가 없어지는 경우가 발생한다. 그 결과 변수 자신이 필요하지 않게 되는 경우가 있다. 변수를 참조하는 문이 있어도 다음과 같은 경우에는

그 변수가 불필요하다. (그림 6)의 (a)와 같은 프로그램에서 x=0을 주어 부분 계산한 것이 (그림 6)의 (b)이다. 변수 y는 재정의로만 참조되어 계산을 하고 있을 뿐이므로 변수 y가 불필요하다는 것을 알 수 있다. 변수가 불필요한 경우는 다음과 같다.

①참조할 문이 없다.

②자신의 수정에만 참조된다.

이들은 일반적으로 프로그램 의존 그래프의 도달성을 조사한 것에 의해서 분석할 수 있다.

```
(a) 프로그램
1  i = 1; y = 1;
2  while i < N do
3        y = y * i;
4        i = i + 1;
5  end;
6  if x = 1 then print(y);

(b) 부분계산 프로그램 (x=0)
1  i = 1; y = 1;
2  while i < N do
3        y = y * i;
4        i = i + 1;
5  end
```

(그림 6) 변수가 불필요하게 되는 예
(Fig. 6) Example that Variable become useless

4.5 효율성 향상을 위한 프로그램의 간소화 방법

4.5.1 간소화 사례

프로그램을 간소화하기 위해 C언어로 기술된 wc 프로그램을 이용하였다[7]. wc는 행수, 단어수와 문자수를 계산하는 프로그램이고, 인수에는 옵션과 화일을 둔다. 옵션은 1, w, c, 3 종류로 각각 행수, 단어수, 문자수의 계산을 행하는 것을 지정한다. 옵션을 생략한 경우는 모든 옵션을 지정한 것이 된다. 다음 (그림 7)은 wc의 소스 코드 일부를 나타낸 것이다. 대역변수 doline, doword, dochar는 각각 행수, 단어수, 문자수의 계산을 행한 경우는 1, 행하지 않은 경우는 0이 된다. 함수 cnt는 실제로 화일의 행수, 단어수, 문자수를 계산하고 출력할 함수이고 메인 함수에서 호출된다. 다음은 wc에서 행수와 단어수만을 계산하는 프로그램 wc를 작성하는 경우, 입력제한 간소화와 출력제한 간소화에 대해서 기술한다.

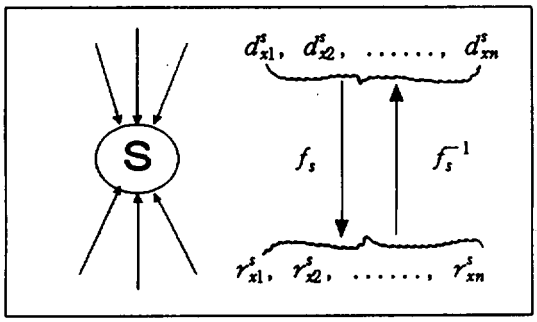
```
1  static int    doline, doword, dochar;
2
3  cnt(file)
4    char  *file;
5  {
6    register u_char *C;
7    register short gotsp;
8    register int len;
9    register long linect, wordct, charct;
10   int fd;
11   u_char buf[MAXBSIZE];
12
13   linect = wordct = charct = 0;
14   if (file)
15      if ((fd=open(file, O_RDONLY, 0)) < 0) {
16                perror(file);
17                exit(1);
18      }
19   else
20      fd = 0;
21
22   for (gotsp=1;len=read(fd, buf, MAXBSIZE);) {
23      if (len == -1) {
24                perror(file);
25                exit(1);
26      }
27      charct += len;
28      for (C = buf; len--; ++c)
29                switch(*C) {
30                   case   NL;
31                            ++linect;
32                   case   TAB;
33                   case   SPACE;
34                            gotsp = 1;
35                            continue;
36                   default;
37                            if (gotsp) {
38                                     gotsp = 0;
39                                     ++wordct;
40                            }
41                }
42      }
43   if (doline)
44      printf(" %7ld", linect);
45   if (doword)
46      printf(" %7ld", wordct);
47   if (dochar)
48      printf(" %7ld", charct);
49   close(fd);
50 }
```

(그림 7) 프로그램 wc의 소스 코드
(Fig. 7) Source Code of Program wc

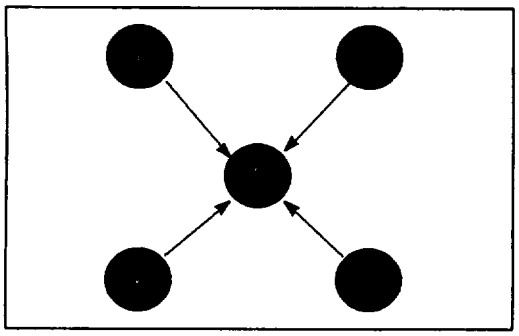
4.5.2 간소화 방법의 구조

지금까지 몇 가지의 프로그램 간소화 방법을 통일한 구조에 대해 기술할 경우 우선 조작 대상으로 간단한 프로그램 의존 그래프를 생각한다. 그래프상의 각 정점을 원래 프로그램 텍스트상의 각 대입문이 if 문 조건식에 대응하고 있다. 이 if문 조건식에 대응하고 있는 정점에서 제어 의존의 에지가 나오고 있고 그 조건식의 값에 따라 제어 의존하고 있는 정점이

실행되거나 되지 않기도 한다. 다른 구조의 문, 예를 들어 while, for, switch문 등은 if 구조에서 표현되는 것으로 본다. 함수 호출문은 대입문 우변이나 조건식 중에 나타날 수 있다. 여기서 (그림 8)과 같은 프로그램 의존 그래프 중의 데이타 의존관계에 착안한다. 위 정점 s에 대응하는 문(대입문 또는 조건식)은 각 변수 $x_1,...,x_n$의 정의역 $d^s_{x_1},...,d^s_{x_n}$에 함수 f_s로 변환한다. 여기서 f_s는 직접 s의 문의 작용을 하는 함수가 아니라 각 정의역 d를 인수로서 각 치역 r에서 변환하는 함수이다 (각 형에 따라 주어진 영역의 파워셋을 인수나 값으로 하는 함수) f_s^{-1}은 f_s의 역함수이다. 일반적으로 최소의 d, r을 부여하는 f_s, f_s^{-1}은 쉽게 정의할 수는 없지만 특정 정수 대입문이나 변수값이 몇 개 밖에 없는 경우, 반대로 특정한 값을 취하지 않는 경우 등은 그들의 제한적인 정보를 함수로서 부여하는 것이 가능하다. 이들 정보가 없는 경우는 형이 허락하는 모든 값을 주도록 한다.



(그림 8) 프로그램 그래프의 데이타 의존 관계
(Fig. 8) Data Dependency Relation of Program Graph



(그림 9) 데이타 의존관계의 예
(Fig. 9) Example of Data Dependency Relation

이와 같이 각각의 문 d와 r의 관계가 함수로 주어짐으로써 다음의 그래프상에서 데이타 의존 관계를 생각할 수 있다. 현재는 간단하기 때문에 (그림 9)와 같은 그래프를 생각한다. 이 경우 p, q로 정의되고 있는 값이 어느 것이든 s에 도달한다. 따라서 p, q에서의 값 영역의 제한은 s에 전해진다. 반대로 s에서 값의 제한은 p, q에도 영향을 준다.

따라서, $d^s_{Xi} \leftarrow (r^p_{Xi} \cap r^q_{Xi}) \cap d^s_{Xi}$ (1)

$r^p_{Xi} \leftarrow r^p_{Xi} \cap d^s_{Xi}$ (2)

$r^q_{Xi} \leftarrow r^q_{Xi} \cap d^s_{Xi}$ (3)

라는 조작을 이용하여 서로의 제한을 반영시킴으로써

$$r^p_{Xi} \cup r^q_{Xi} = d^s_{Xi} \quad \text{(a)}$$

로 할 수가 있다. 마찬가지로,

$r^s_{Xi} = (d^u_{Xi} \cup d^u_{Xi}) \cap r^s_{Xi}$ (4)

$d^u_{Xi} \leftarrow d^u_{Xi} \cap r^s_{Xi}$ (5)

에 의해, $r^s_{Xi} = d^u_{Xi}$ (b)

로 할 수가 있다. 그래프의 각 정점에 대해서 d, r의 값을 구한다. 우선 각각 d, r의 초기값은 식의 값이 정해질 때에는 그 값을 요소로 하는 부분 영역, 그 이외는 그 식이나 변수의 형이 주어진 영역 전체로 한다. 그리고 그래프의 모든 정점과 모든 변수에 대해서 (a), (b)가 성립함으로 f_s, f_s^{-1}를 사용하여 (1)에서 (b)의 조작을 실행하며 차례로 d, r을 작게 해 간다.

변화가 없어지고 어느 장소에서도 (a), (b)가 성립한 때의 그래프를 생각한다. s가 대입문 x_i, :=..일 때, $r^s_{Xi} = 0$이 되는 경우 그 정점은 불필요하므로 삭제할 수 있다(부작용이 있을 때는 그 부작용을 준 변수에 대해서도 0이 될 필요가 있을 것이다). if문의 조건문이 참 또는 거짓이 되도록 한 각 변수의 d가 주어진 때는 제어 의존의 에지를 더듬어 가서 불필요한 정점을 삭제할 수 있다. 여기에서 입력 제한에 대해 생각해보면 이것은 입력문이 판독하는 값의 영역을 제한하는 것에 해당한다. 입력 변수의 하나를 고정하여 부분 계산하는 것은 그 변수의 영역을 형이 주어진 영역 전체에서 그 하나의 값으로 표현할 수 있다. 입력 변환 하나를 삭제하는 것을 그 변수의 영역을 0으

로 하는 것으로 표현된다. 다음에 출력 제한에 대해 생각해보면 필요없는 출력변수의 영역을 0으로 하는 것으로 필요한 출력변수가 의존하는 문, 즉 슬라이스를 남길 수가 있다. 이와 같이 부분 계산, 슬라이스 등의 정적인 조작을 할 수가 있다. 여기서는 입출력에만 제한을 주어 다른 곳으로의 파급을 조사하는 방법만을 기술했지만 도중 특정한 문에 착안하여 그곳의 영역을 제한하는 것에 의한 프로그램의 간소화도 생각할 수 있다.

5. 효율성의 측정과 평가 사례

효율성 측정을 위한 대상 업무는 D 증권에서 개발하고 있는 프로젝트로서 주요 목적은 국내외 자본 시장의 환경변화에 적극적으로 대처하고 신속한 업무처리를 통한 대고객 서비스 향상과 경쟁력을 확보하는데 있다. 따라서 본 절에서는 위와 같은 목적에 부합하면서 효율성에 관한 내부특성 메트릭스의 타당성을 입증하기 위하여 개발 소스 코드를 이용하여 품질을 측정하고 평가한 사례를 기술하였다.

5.1 평가 대상 업무의 구성

개발중인 프로젝트는 크게 시스템 업무와 개발업무 두가지로 분류되어 있으며 시스템 업무는 정보시스템, 대외접속 시스템, 비상대기 시스템, 업무계 시스템, 업무처리 시스템, 영업점 시스템에 대한 품질측정과 평가를 수행한다. 개발업무에 대해서는 출납업무, 매매업무, 공통업무, 권리업무, 청약업무 등에 대한 효율성의 품질 측정과 평가를 실행하였다.

5.2 업무별 평가 결과

평가 대상 업무가 증권업무에 대한 통합적이고 고신뢰성을 요구하는 시스템인 만큼 본 연구에서 제안한 메트릭스와 측정표 즉, 범용적인 시스템 품질평가 요소들뿐만 아니라 개발 목적에 적합한 평가요소들을 추가하여 평가하였다. 본 절에서는 D 증권 프로젝트에 대해 각 업무별로 효율성 품질측정 결과를 기술하고 문제점과 해결 방안을 예시하였다.

〈표 7〉은 실행효율성 메트릭스에 대한 부특성과 내부특성의 평가 결과를 나타낸 것으로 출납 업무의 실행효율성에 관한 부특성 메트릭스의 경우, IOpt가 5점으로 최고점을 나타내고 있으며 Tt가 71.8점으로 최하점을 나타내고 있다. 전체적으로는 청약업무가 평균 78.9로 최고점을 나타내고 있으며 본 연구에서 정한 기준에 따르면 '보통' 수준의 측정 결과를 보이고 있다.

〈표 7〉 업무별 실행효율성의 측정결과
〈Table 7〉 Measurement Result of Execution Efficiency

특성	메트릭스		출납	매매	공통	권리	청약	결과
실행효율성	부특성	Tt	71.8	75.8	79.2	81.3	81.4	77.9
		Rt	81.3	84.5	73.6	78.4	73.4	78.2
		Ct	73.6	82.5	87.6	78.3	73.5	79.1
		CRt	73.9	81.4	72.6	72.5	83.4	76.8
		IOpt	83.5	73.5	79.4	82.4	73.7	78.5
		Wt	73.6	77.2	78.5	75.3	82.4	77.4
		NPt	73.9	71.8	77.4	73.6	82.6	75.9
		TPt	72.8	81.9	73.7	71.4	81.4	76.2
		Pa	75.7	72.7	74.6	73.5	78.4	74.9
		TPn	73.4	72.4	74.6	81.4	79.4	76.2
		평균	75.4	77.4	77.1	76.8	78.9	77.1
	내부특성	PDi1	81.5	71.7	72.5	82.5	81.6	77.9
		PDi2	81.5	72.4	83.4	71.5	78.2	77.4
		PDi3	72.6	82.4	71.6	71.8	78.3	75.3
		PDi4	81.4	71.5	79.2	87.2	88.2	81.5
		PDi5	85.1	78.1	81.4	82.5	77.9	81.0
		평균	80.4	75.2	77.6	79.1	80.8	78.6

〈표 8〉은 자원효율성의 메트릭스에 대한 부특성과 내부특성의 평가 결과를 나타낸 것이다. 출납업무의 자원효율성에 관련된 부특성의 메트릭스의 경우, RMa가 84.3점으로 최고점을 나타내고 있으며 CPUa가 72.3점으로 최하점을 기록하고 있다. 전체적으로는 매매업무가 평균 79.7로 최고점을 나타내고 있으나 이 등급은 '보통' 수준의 측정 결과이다.

〈표 9〉는 실행효율성과 자원효율성에 관한 측정표를 이용하여 품질부특성과 내부특성의 값을 측정한 결과를 집계한 것으로 전체업무의 평균 점수가 71 80점 사이로 '보통' 수준에 머무르고 있음을 알 수 있다. 본 연구에서는 80점을 합격수준으로 정하고 수준에 미달되는 특성에 대해서는 문제점과 개선방안을 예시하여 품질향상을 도모할 수 있도록 하였다. 품질평가를 통해 전체적으로 품질수준이 다소 미흡하다는 것을 알 수 있으며 앞으로 반복적인 평가를 통해 평

가 기준의 타당성을 검증해야 할 필요성이 요구된다.

〈표 8〉 업무별 자원효율성의 측정 결과
〈Table 8〉 Measurement Result of Resource Efficiency

특성	메트릭스		출납	매매	공통	권리	청약	결과
자원효율성	부특성	CPUa	72.3	76.6	79.3	78.3	72.6	75.8
		MAa	73.5	77.3	79.3	73.5	84.4	77.6
		RMa	84.3	84.8	77.3	71.5	72.5	78.1
		VMa	76.7	75.6	76.4	83.3	84.4	79.3
		WSv	74.6	78.3	81.5	73.6	79.4	77.5
		FUa	82.5	72.5	73.5	81.4	79.3	77.8
		Na	78.4	82.5	72.4	78.3	73.6	77.0
		Mr	73.5	82.6	74.6	75.7	82.5	77.8
		Cr	73.6	81.2	82.5	81.8	78.2	79.5
		Fr	78.4	81.5	71.3	79.4	73.5	76.8
		Nr	72.5	83.7	75.6	71.4	82.4	77.1
		평균	76.4	79.7	76.7	77.1	78.4	77.7
	내부특성	RDi1	82.4	81.3	73.4	71.9	79.2	77.6
		RDi2	77.2	79.2	81.9	81.4	72.5	78.4
		RDi3	76.6	82.4	81.8	82.7	78.2	80.3
		RDi4	71.7	78.3	72.5	87.2	79.2	77.8
		평균	76.9	80.3	77.4	80.8	77.3	78.5

〈표 9〉 효율성의 최종 평가 결과
〈Table 9〉 Final Evaluation Result of Efficiency

특성		출납	매매	공통	권리	청약
부특성	실행효율성	75.4	77.4	77.1	76.8	78.9
	자원효율성	76.4	79.7	76.7	77.1	78.4
내부특성	실행효율성	80.4	75.2	77.6	79.1	80.8
	자원효율성	76.9	80.3	77.4	80.8	77.3
평균		77.3	78.2	77.2	78.5	78.9
평정		보통	보통	보통	보통	보통

5.3 문제점과 개선방안

제안한 내부특성의 메트릭스에 따라 업무별 효율성을 측정한 결과 각 품질특성별, 업무별 평가 결과가 전체적으로 합격수준으로 설정했던 80점에 미달되는 저수준임을 알 수 있다. 품질평가는 결과를 산출하는 것 뿐만 아니라 결과에 대한 문제점을 분석하여 개발자에게 제시함으로써 품질을 향상시키는 것이 목적이므로 〈표 10〉과 같이 내부특성의 메트릭스에 관련해서 평가 대상 소프트웨어에 나타난 문제점과 개선방안을 예시하였다.

〈표 10〉 문제점과 개선 방안의 일부
〈Table 10〉 A Part of Problems and Progress Plans

내부특성	요인항목	세부항목(문제점)	개 선 방 안
실행효율성	입출력 회수	외부장치와의 입출력 회수와 외부대 이타의 입출력 회수가 필요 이상으로 많다.	공통성이 있는 자료는 한번에 입·출력하여 프로그램에서 입출력 회수를 최소화 한다.
	동적 스텝수	CPU로 처리되는 동적 스텝(계산식)이 최적화되어 있지 않다.	동일한 계산식이 반복되어 나타나는 경우 미리 계산한 후 계산결과를 사용하는 식으로 변환하여 동적 스텝수를 줄인다.
	평균 기동시간	처리 과정중 사용되지 않는 기능들이 존재한다.	사용되지 않는 기능을 제거하거나, 필요한 기능인 경우에는 사용될 수 있도록 수정한다.
	⋮	⋮	⋮
자원효율성	데이타의 중복률	데이타를 통합하여 사용하지 않아 동일 데이타가 중보하여 나타난다.	데이타베이스의 조건을 충족할 수 있도록 가능한한 중복성을 배제하여 일치성(consistency)을 보장하고 기억장소를 낭비하지 않도록 수정한다.
	출력 페이지 밀도	출력되는 페이지당 정보량이 필요이상 많거나 적은 경우가 많다.	인간의 단기기억의 한계는 7±2개이므로 한 페이지당 지나치게 많은 정보를 포함하지 않도록 한다.
	⋮	⋮	⋮

6. 결 론

소프트웨어의 다기능화와 대규모화가 진전함에 따라 사용자의 요구를 어느 정도 충족시킬 수 있으나 성능이 저하됨으로써 고성능의 시스템을 새로이 도입해야 하는 경우가 발생하고 있다. 반면, 사용자는 다기능 소프트웨어의 극히 일부 기능만을 사용하는 경우가 대부분이기 때문에 시스템의 효율성을 저하시키는 결과를 가져왔으며, 사용자의 입장에서는 필요 이상의 대규모 시스템을 도입하여 사용함으로써 상대적으로 도입 비용 및 운영 비용의 낭비를 초래하게 된다.

따라서, 본 연구에서는 품질특성인 효율성을 실행효율성과 자원효율성으로 나누어 품질부특성으로 구성하였으며 개발자의 관점인 품질 내부특성을 정의하여 소프트웨어의 내부적인 특성을 개발 산출물을 통하여 측정할 수 있도록 하였다. 또한 품질부특성과 내부특성에 대해 실행효율성과 자원효율성의 관점에

서 품질평가 메트릭스를 정의하고 내부특성의 메트릭스에 대해서는 측정표를 구성하여 용이하게 품질평가를 할 수 있도록 하였다. 그리고 소프트웨어의 효율성을 향상시키기 위한 방안으로는 입출력 영역 제한에 의한 프로그램의 변환에 착안한 간소화 방법에 대해서도 기술하였다. 또한 효율성 평가 메트릭스를 품질부특성과 내부특성의 관점에서 소프트웨어 개발 산출물의 품질평가에 적용하여 측정 결과를 예시하고 문제점 및 개선방안을 기술함으로써 피드백을 통한 품질향상을 도모하였다.

향후 연구 과제로 효율성 관련 메트릭스를 이용하여 품질평가를 반복실시함으로써 문제점을 발견하고 개선하는 과정을 통해 메트릭스를 검증해야 하며 또한 메트릭스의 값을 구하는 과정에서 주어지는 가중치에 대한 객관성에 대해서도 연구가 병행되어야 할 것이다.

참 고 문 헌

[1] S. Horowitz and T. Rops, "The Use of Program Dependence Graphs in Software Engineering," Proc. 14th International Conference on software Engineering, Melbourne, Australia, pp. 392-411, May 1992.

[2] Uwe Meyer, "Techniques for Partial Evaluation of Imperative Languages," Proc. Symposium on Parial Evaluation and Semantics-Based Program Manipulation, pp. 94-105, Jun 1991.

[3] Thadhani, A. J., "Factors Affecting Programmer Productivity during Application Development," IBM Systems Journal, Vol. 23, No. 1, pp. 19-35, 1984.

[4] Ohba, M., "Software Quality = Test Accuracy Test Coverage," Proc. of 6th ICSE Tokpo, IEEE, pp. 287-293, 1982.

[5] Roger S. Pressman, *Software Engineering*, 3rd. Ed., McGraw-Hill International Editions, 1992.

[6] Brian W. Kernighan, *The C Programming Language*, 2nd. Ed., Prentice Hall, 1988.

[7] Zuse, H, *Software Complexity: Measure and Methods*, New York, Walter de Grutyer, 1991.

[8] Lowell Jay Arthur, *Measuring Programmer Productivity and Software Quality*, John Wiley & Sons, Inc., 1985.

[9] K., John Gough, *Syntax Analysis and Software Tools*, Addison Wesley, 1988.

[10] L. A. Belady, M, M. Lehman, "A model of large program development," IBM System Journal, Vol. 15, No. 3, pp. 225-252, 1976.

[11] ISO/IEC 9126, Information technology-Software Product Evaluation-Quality Characteristics and Guidelines for Their Use, ISO, 1991.

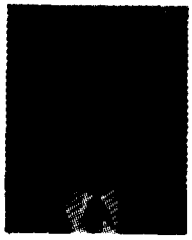
[12] 菅野文友, "ソフトウエア品質管理事例集," 日科技連, 1990.

[13] 森口, "ソフトウエア品質管理ガイドブック," 日本規格協會, 1990.

[14] 정호원, 양해술, ISO 9000 시리즈와 소프트웨어 품질시스템(上), 하이테크정보, 1993. 3.

[15] 양해술, 소프트웨어의 품질평가 체계 및 자동화 도구의 개발, 한국통신 장기기초연구과제, 최종보고서, 1995.

[16] 양해술, 품질관리 방법론 및 지원도구의 개발, 과학기술처 STEP2000 지원과제 제3차년도 최종보고서, 1997.



양 해 술

1975년 홍익대학교 공과대학 전기공학과 졸업(학사)
1878년 성균관대학교 정보처리학과(석사)
1991년 日本 오사카대학교 기초공학부 정보공학과 S/W공학 전공(공학박사)
1975년~1979년 육군중앙경리단 시스템분석장교
1986년~1987년 日本 오사카대학교 객원연구원
1980년~1995년 강원대학교 전자계산학과 교수
1993년~1994년 한국정보과학회 학회지 편집부위원장
1994년~1995년 한국정보처리학회 논문지편집위원장
1994년~현재 한국산업표준원(KISI) 이사
1995년~현재 한국소프트웨어품질연구소(INSQ) 소장
관심분야: 소프트웨어공학(특히, S/W 품질보증과 품질평가, 품질감리, 품질컨설팅, OOA/OOD/OOP, CASE, SI), 소프트웨어 프로젝트관리