재사용 소프트웨어 품질평가 도구 개발

최 은 만† · 남 윤 석††

요 약

소프트웨어를 재사용하는 경우 부품의 품질은 새로 개발한 시스템의 품질에 매우 큰 영향을 미친다. 따라서 재사용 라이브러리를 구성할 때 품질에 대한 평가와 검증은 필수적이다. 여러번 다시 사용하는 재사용 소프트웨어는 일반적인 일회적 소프트웨어에 대한 품질 평가와는 다른 기준으로 평가할 필요가 있다. 사용자 인터페이스나 기능적인 측면보다는 확장성과 정확성 등이 더욱 중요한 품질 요소가 된다. 본 논문은 멀티미디어 재사용 부품 저장소 및 재사용 시스템(Reusable Software for Multimedia Objects)의 일부인 객체 부품화 및 품질 평가 기술 개발에 관한 것으로 C++ 및 IDL(Interface Definition Language)로 표현된 재사용 부품을 읽어 구문 형식, 구조, 객체 결합도, 객체 응집도, 객체 복잡도, 이해도 등의 관점에서 분석하고 품질 만족도를 출력하는 품질평가 도구를 다루었다. 또한 분산 환경에서 품질 검증 시스템을 사용할 수 있도록 CORBA기반에서 설계하였다.

Development of a Quality Assessment Tool for Software Reuse

Eun Man Choi† · Yoon Suk Nam††

ABSTRACT

Quality of a new system is closly related to the quality of components in reuse repository. Quality assessment is essential to construct a reuse library. Definition of quality and method of assessment are totally different in reuse environments. User interface, functionality, performance are main factor in non-reuse development environment. However, reuse environment needs more reusability, extensibility, generality, and maintainability in quality assessment. This paper describes a development of quality assessment tool for multimedia object reuse conponents. Tool gets reuse components described by C++ or IDL, and analyses style, structure, coupling, strength, complexity, understandability, etc. Ultimatly the tool generates quality satisfaction degree for reuse programmers. Quality assessment services are supported in distributed object architecture, CORBA.

1. 서 론

소프트웨어 재사용 기술은 오래 전부터 소프트웨어 개발의 생산성과 품질을 향상시키기 위한 방안으로 제시되어 왔다[5][6][7]. 소프트웨어를 마치 하드웨어의 부품처럼 재사용한다면 소프트웨어 개발 속도는 매우 빨라질 것이며 소프트웨어 위기 현상을 해결할 것이라는 전망이 있어 왔다. 그러나 현실 세계의 소프트웨어 개발은 재사용이 보편화 되어 있지 않다.

재사용 부품을 분류하고 저장하여 재사용자가 요구하는 부품을 찾아 새로운 시스템으로 합성하는 기능이 완벽히 지원되지 않기 때문이다. 그 중에서도 부품을 저장할 때 재사용 부품을 품질 측면에서 평가하는 기능은 제대로 확립되어 있지 않고 상용화된 제

† 정 회 원:동국대학교 정보산업학부 교수
†† 정 회 원:한국전자통신연구원 연구원
논문접수:1997년 2월 24일, 심사완료:1997년 6월 25일

품은 거의 없다. 재사용 부품은 이미 한번 이상 사용되어 그 품질을 인정받은 제품이나 재사용 라이브러리의 부품 품질 기준에 맞는지 확인할 필요가 있다. 또한 품질과 관련된 여러 가지 측정 결과로 부품의 특성을 알아낼 수 있다[9].

객체지향 프로그램에 관한 메트릭이 많이 제안되어 있으나 다음과 같은 문제점들이 존재한다.

- 객체지향 프로그램은 설계 구조가 중요함에도 불구하고 대부분의 품질 메트릭은 복잡도나 크기, 단순한 구문 중심의 메트릭을 사용한다.
- 품질에 대한 평가보다 단순한 통계 중심 메트릭에 불과하다.
- 부품의 구성과 품질에 대한 기준이 미흡하다.

이 논문에서는 위와 같은 문제점에 대하여 다음과 같은 해결 방법을 제시하는 것이다.

- 객체지향 기본 메트릭을 포함한 객체지향 특유의 품질 메트릭에 대한 제안하고 구현한다.
- 객체지향 부품의 품질을 구문, 스타일, 구조, 패키지화 등 다양한 측면으로 분석하여 평가한다.
- 재사용 부품의 구성과 품질에 대한 기준을 제시한다.

재사용 부품이 분산 객체 환경에서 제공된다면 부품의 품질도 CORBA와 같은 환경에서 평가할 수 있는 환경이 되어야 한다. 따라서 객체 부품의 평가 서비스도 OMA 구조에서 구현될 필요가 있다. 품질 평가 도구는 C++ 및 IDL로 표현된 부품을 읽어 구문, 형식, 구조, 인터페이스 등의 관점에서 분석하고 메트릭 결과를 출력한다. 도구의 구현은 OMA 분산 환경에서 이용할 수 있도록 서버로 지원한다.

객체 부품 품질 측정 도구의 설계는 Rumbough가 제안한 OMT 방법을 이용하여 표현하였다. 전체적인 시스템의 요소를 파악하여 구조를 정하고 필요한 객체와 객체들의 관계를 객체 다이어그램으로 나타내었다. 또한 시스템의 사용자 인터페이스를 제시하고 사용자들과의 상호대화를 시나리오로 작성하고 이를 동적 모형으로 표현하였다. 사용자와의 상호작용이 없는 구체적인 메트릭 값을 구하는 작업은 자료 흐름도로 표현하였다.

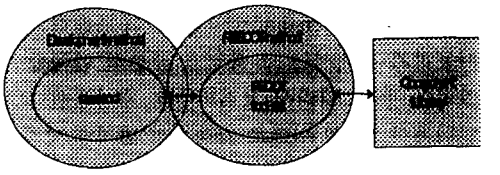
2. 관련 연구

2.1 REBOOT

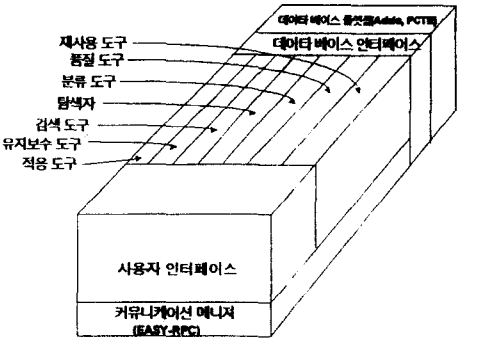
REBOOT(REuse Based on Object Oriented Techniques)는 1990년 9월부터 4년간 124 man-years을 투입한 재사용 시스템이다. 이 시스템은 기존의 CASE method를 그대로 이용할 수 있으며 재사용을 위한 개발(develop for reuse)과 재사용에 의한 개발(develop with reuse)을 지원하며, 역공학 도구, 품질 도구, 분류 도구, 검색 도구, 유지보수 도구 등의 도구 세트를 포함한 재사용을 위한 통합도구이다.

REBOOT의 운영환경은 Sun과 DFX20이며 소프트웨어 개방환경 프레임워크의 표준인 ECMA에서 컴파일 된다. 또한 REBOOT 도구는 다음을 포함하고 있다.

- OSE /MOTIF 상에서 동일한 사용자 인터페이스를 구축
- 커뮤니케이션매니저와 같은 소프트벤치를 통한 통신



(그림 1) REBOOT와 다른 시스템과의 관계
(Fig. 1) Relationships of REBOOT to other systems



(그림 2) REBOOT 시스템 구조도
(Fig. 2) System architecture of REBOOT

- 다양한 데이타베이스 플랫폼상의 상위에서 실행되는 고수준 데이타 베이스 인터페이스를 통한 데이타 공유

이중에서 품질 도구는 이식성, 융통성, 이해성, 신뢰성 등의 4가지 요소에 대한 사전 품질평가와 사후 품질평가를 지원한다. 사전 품질평가는 재사용 부품을 재사용 시스템에 저장하기 전에 행하는 품질 평가로서 기준 품질에 미치지 않는 부품은 시스템 내에 저장되지 않는다. 이에 반해 사후 품질평가는 실제 재사용을 위해서 경험적인 요소를 가지고 품질평가를 하게 된다. 이외에도 재사용 부품에 있는 재사용 내역을 이용해서 그 부품의 재사용도를 측정할 수 있다. 물론 재사용 내역 정보는 부품검색 때에도 유용하게 사용된다.

REBOOT에서는 재사용성 품질을 다음 네 가지로 측정한다.

① 호환성(portability): 다른 컴퓨터 시스템 환경으로 부품이 전환되기 쉬운정도.

② 유연성(flexibility): 다른 기능적 요구를 위하여 부품을 개작하거나 변경하여 다른 환경에서 사용하기 용이한 정도.

③ 이해용이성(understanderbility): 소프트웨어를 재사용하기 위해서는 그 목적을 충분히 이해할 수 있어야 한다. 따라서 블랙박스로 재사용한다면, 인터페이스만의 이해로 가능하며 변경후 재사용한다면(화이트박스 재사용) 구현 내용도 이해해야 한다.

④ 적합성(confidence): 모듈, 프로그램, 시스템이 원래 구축된 환경이 아닌 다른 환경에서 정의된 목적을 만족시킬 주관적 확률.

적합성은 신뢰도와 매우 밀접한 관련을 갖지만, 같다고는 할 수는 없다. 신뢰성은 주어진 시간에 정상적 기능을 할 수 있는 확률을 말하며, 적합성이란 다른 환경에서 재사용할 때 정확하게 동작할 확률을 말한다[1].

REBOOT는 위와 같은 네가지 측면의 제품 메트릭을 라이브러리에 등록할 때와 등록하여 재사용하면서 그 경험이 축적되었을 때 다시 한 번 측정한다[8].

2.2 QUALMS

QUALMS(QUality Analysis and Metrication Software)는 소프트웨어에 관한 면밀한 분석과 품질 평가하는 도구이다[11]. 구조 메트릭이나 복잡도 메트릭은 prime flowgraph에 바탕을 두고 있다. QUALMS가 측정하는 메트릭은 다음과 같은 19개의 메트릭이다.

- 단순한 구조 특성 측정 메트릭: prime flowgraph의 길이, 가장 큰 prime, number of occurrences of named primes, 중첩의 깊이, 평균레벨, 경로 수
- 결합한 구조 특성 측정 메트릭: McCabe, Prather, 간단한 경로의 수, Basili Hutchens Syntactic Complexity
- 복잡한 복잡도 특정 메트릭: Lambda, YAM, VINAP
- 테스팅 메트릭: 가능 경로 테스팅, 문장 테스팅, 각 루프를 방문된 수

2.3 ESCORT

강원대학교와 한국소프트웨어 품질연구소에서 공동으로 연구한 소프트웨어 품질평가 도구로서 C/C++ 언어로 작성된 원시 프로그램에 정적분석을 행하며, 원시 프로그램을 수정하고 평가하여 소프트웨어의 품질 향상을 도모하는 것을 목적으로 한다[12].

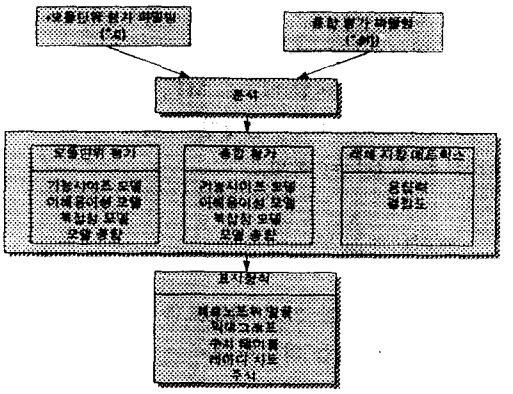
또한 ESCORT는 소프트웨어 부품의 보수성 관점에 기인하여, 원시 프로그램을 측정하는 대상에 관해서 다음과 같은 특징을 가지고 있다.

첫째, 보수성 있는 소프트웨어 부품의 품질 향상을 도모하며, 둘째, 기계적, 자동적으로 측정하며, 셋째, 객관적으로 평가하며, 넷째, 처리결과에 따른 품질개선 방안을 제시하며, 다섯째, 다양한 모듈과 부품 작성 체제를 정립할 수 있다.

(그림 3)은 ESCORT 시스템의 구조도이다.

현재 ESCORT는 명세서 품질평가 도구 ESCORT-S (Specification)와 분석단계에서의 품질평가 도구 ESCORT-A(Analysis), 설계단계의 품질평가 도구 ESCORT-D(Design), C와 C++언어로 기술되어 있는 소스코드의 품질을 측정하는 ESCORT-C(coding) 그리고 유지보수단계에서 이용 가능한 오류복구 지원 도구 ESCORT-M(Maintenance)으로 분류하여 체제와 도구의 설계 및 개발을 진행하고 있다.

보수가 용이한 소프트웨어를 개발하기 위해서는 소프트웨어를 품질을 정량적으로 평가하기 위한 모



(그림 3) ESCORT 시스템의 구조도
(Fig. 3) System architecture of ESCORT

델이 필요한 데 ESCORT에서는 기본 모델로 기능 사이즈 모델, 이해용이성 모델, 복잡성 모델, 구조화 모델을 제시하여 보수성 있는 소프트웨어의 품질을 평가한다. 각 모델에 포함된 요인 항목은 정량적인 수식에 의해 결과가 도출되며, 각각의 요인 항목에 대해서는 결과 값의 하한을 정의하기 위한 허용치와 품질의 우수성 정도를 나타내는 표준치를 두어 소프트웨어가 사용되는 영역과 특성을 고려하여 결과값에 대한 판단을 가능하도록 하고, 소프트웨어 전문가에 의해 정의될 수 있도록 한다.

2.3.1 기능 사이즈 모델

모듈 하나를 유지하는 기능의 수를 측정하여 정량화 한 모델이다. 원시 프로그램의 각각의 모듈이 유지하고 있는 기능은 작고, 모듈당 한 개의 기능이 바람직하다는 점을 고려한 것이다. 기본적인 요인항목으로는 출구수, 내부면 수의 유효범위 블럭수, 명령어 카테고리 수, 함수군의 수, 최대 블럭의 수, 두 번째 블럭의 수로 구성한다.

2.3.2 이해용이성 모델

원시 프로그램의 이해용이성을 정량화한 모델이다. 작성된 원시 프로그램이 시각적으로 이해하기 쉽다면 프로그램을 빠르게 보수할 수 있다는 생각을 기초로 한다.

기본적인 요인항목으로는 주석, 주석 위치, 함수의 주석 수, 변수의 유효범위 평균치, 조건문의 평균 블록길이, 스텝수, 전역변수의 밀도, 출구 수, goto문의 수로 구성된다.

2.3.3 복잡성 모델

논리의 복잡성을 정량화한 모델이다. 원시 프로그램의 논리가 단순 명료한 것이 바람직하다는 점을 고려한 것이다. 기본적인 요인항목으로는 실행문 수, Halstead의 어휘 사이즈, Halstead의 프로그램 길이, Halstead의 프로그램 볼륨, McCabe의 척도, 비트 정보량으로 구성된다.

2.3.4 구조화 모델

모델의 결합 복잡성을 정량화한 모델이다. 다수의 모듈로 구성되어 있는 소프트웨어에서 모듈 사이의 관련성을 단순 명쾌하게 유지하도록 한 것이다. 기본적인 요인항목으로는 전역변수의 사용정도, 모듈당 출구수, 타 모듈의 호출 정도, 호출하는 수의 표준편차로 구성된다.

3. 품질과 메트릭

3.1 소프트웨어 품질 특성

재사용 소프트웨어의 품질 특성은 아래와 같이 재사용성, 이해 용이성, 이식성, 유지 보수성, 신뢰성 등의 5가지 특성으로 분류하였다.

3.1.1 재사용성

재사용 부품의 멤버 함수의 캡슐화가 잘 되어 있는지의 여부, 각각 부품들간의 상속 관계는 잘 유지되는지, 부품 내부의 다형성은 잘 되어 있는지를 측정한다. 재사용성은 재사용 부품의 복잡도, 상속 트리의 깊이, 상속 트리의 넓이, 객체간의 결합도와 밀접한 관계가 있다.

3.1.2 이해 용이성

재사용 부품의 논리적 개념과 응용을 사용자가 얼마나 잘 이해할 수 있는가, 사용자가 그 재사용 부품을 얼마나 잘 조절할 수 있고 빨리 배울 수 있는가에 기준을 두어 측정한다. 이해 용이성은 재사용 부품의 복잡도, 상속 트리의 깊이, 객체간의 결합도와 관계가 깊다[3].

3.1.3 이식성

이식성은 다른 환경에 적응하기 위하여 필요한 행동이 얼마인지, 얼마나 특정한 환경에 설치하기 쉬운지, 다른 소프트웨어에 얼마나 적응 할 수 있는지를 측정한다. 이식성은 3.2.5에 소개된 객체간의 결합도, 반응도, 3.2.6에 기술된 객체간의 응집 결핍성을 사용하여 측정할 수 있다.

3.1.4 유지 보수성

재사용 부품의 유지 보수성은 부품 실행 오류의 원인이나 비효율성을 쉽게 찾고 수정에 의한 예상치 않은 결과의 영향을 얼마나 적게 받는지, 수정된 소프트웨어를 검증하는데 드는 노력이 얼마인지를 측정한다. 유지보수성은 재사용 부품의 복잡도, 상속 트리의 넓이, 객체간의 결합도와 관계가 깊다.

3.1.5 신뢰성

소프트웨어의 결함에 의한 실패의 정도, 소프트웨어의 결함에 대해서 안정화를 유지할 수 있는 정도, 결함에 의해 영향을 받은 데이타를 복구하고 그 성능을 유지하는 정도를 재사용 부품의 신뢰도 평가 기준으로 한다. 신뢰도는 재사용 부품의 복잡도가 클 경우 결함에 복구할 수 있는 정도가 적어진다.

3.2 품질 메트릭

품질 메트릭은 아래와 같이 8가지로 분류하였다.

3.2.1 자식노드의 수 : Number Of Children(NOC)

NOC는 클래스 계층구조안에서 클래스에 종속적인 서브 클래스들의 개수를 말한다. 즉 해당 클래스가 직접적으로 상속받는 클래스의 개수를 말한다. 이 수치가 클수록 좋으나 너무 클 경우에 이 클래스에 영향을 받는 클래스가 많다는 뜻이므로 적당한 것이 좋다.

- 많은 수의 서브클래스들은 설계상에 있어서 다른 클래스에게 영향을 줄 가능성이 많다.
- 만약 서브 클래스가 많은 서브클래스의 수를 가진다면, 그 클래스 안의 메소드들에 대해 많은 테스팅을 해야만 한다.

3.2.2 클래스당 깊이(가중치)를 가진 메소드의 수 : Weight Method per Class(WMC)

클래스 안에 정의된 메소드 M1 , M2 ... Mn과 함께 클래스 C1, C2, ... Cn이 존재한다. 이때 c1, c2 ... cn을 메소드의 복잡도라 하면,

$$WMC = \sum_{i=0}^{n} c_i$$

여기서 가중치를 가진 메소드란 메소드 자체가 가지는 정적 복잡도를 의미한다.

- 메소드들의 개수와 메소드들의 복잡도는 클래스의 개발과 유지보수에 요구되는 노력과 시간이 어느 정도인가를 예측할 수 있다.
- 클래스 안에서의 많은 메소드들의 개수는 자식 클래스들과 충돌 가능성이 있는데, 자식클래스는 클래스 안에서 정의된 모든 메소드들을 상속받기 때문이다.
- 메소드들과 클래스는 재사용 가능성의 한도 내에서 좀더 응용문제에 대해 구체적이 되려는 경향이 있다.

WMC의 측정의미는 메소드의 수와 메소드 각자가 가지는 복잡성은 객체를 개발, 유지하는 노력과 시간에 비례하는 지시자의 역할을 한다고 볼 수 있다.

3.2.3 상속의 깊이 : Depth of Inheritance Tree (DIT)

클래스에 있어서 상속의 깊이를 클래스에 대한 DIT라 한다. 예를 들면, DIT는 상속트리의 루트 노드로부터의 최대의 길이이다. 따라서 DIT는 얼마나 많은 부모클래스가 현재 클래스에 영향을 주었는지의 척도가 된다. 이것은 상속계층에서 클래스의 깊이가 깊을수록 메소드 수가 많아지므로 좀 더 복잡하게 된다는 것을 의미한다.

- 상속의 깊이가 깊은 클래스는 계층구조 안에서 많은 수의 메소드들이 존재한다는 것을 뜻한다.
- 상속이 깊은 트리들은 좀더 많은 메소드들과 클래스들을 포함하기 때문에 복잡도가 높다.
- 특정 클래스의 깊이는 계층구조 안에서 상속된

메소드들의 재사용 가능성을 높일 수 있다.

3.2.4 지역 메소드의 수 : Number Of local Methods (NOM)

한 클래스에 정의된 모든 메소드의 수를 NOM이라고 한다. 이 메트릭이 너무 커지면 클래스가 복잡해져서 관리하거나 사용하기가 어려워진다.

3.2.5 객체간의 결합도 : Coupling Between Objects (CBO)

하나의 클래스에 대한 CBO는 다른 클래스와의 결합한 수를 측정하는 것인데 이때 상속성의 관계는 제외한다.

- 클래스 사이의 과도한 결합도는 디자인 모델의 손실과 재사용을 방해한다. 그러므로 조금 더 독립적인 클래스는 다른 응용에서 쉽게 재사용 할 수 있다.
- 모듈성을 개선하고 캡슐화를 증진시키기 위하여, 클래스 간의 결합도는 최소화되어야 하며, 결합도가 높을수록 유지보수와 프로그램의 이해가 어려워진다.
- 결합도 메트릭은 다양한 종류의 디자인과 복잡한 테스팅을 나타내는데 있어 유용하다.

3.2.6 메소드에서 응집도의 결핍성 : Lack of COhesion in Method (LCOM)

하나의 클래스에서 임의의 메소드 Mi가 사용하는 인스탄스 변수들의 집합을 (Ii)라 하면 이 클래스에서 메소드 Mi, ... Mn에 대해 n개의 (Ii), ... (In)이 존재한다. 여기서 LCOM은 이 n개 집합들의 교집합으로 구성된 집합의 개수이다.

$$LCOM = |P| - |Q|,\ |P| > |Q| \text{이면}$$
$$= 0, \text{그 외의 경우}$$

여기서 P는 메소드들 안에서 사용된 인스턴스들이 공동으로 사용된 메소드의 수이며 Q는 인스턴스가 전혀 다른 것이 사용된 메소드의 수이다.

이 측정은 각 메소드가 참조하거나 사용하는 인스턴스 변수의 집합을 구한 후 그들의 교집합을 구하고 그들의 교집합에서 공통되지 않은 집합의 수를 구하는 것이다.

- 결합정도가 클수록 현재의 클래스가 2개나 그 이상의 클래스로 나누어 설계되어야 한다.
- 클래스 안의 메소드들의 응집도가 크면 해당 모듈의 캡슐화를 증진시킨다.
- 낮은 응집도는 복잡도를 증가시키며 개발과정의 오류를 증가시킨다[3].

3.2.7 클래스에 대한 반응도 : Response For a Class (RFC)

클래스가 호출하는 모든 메소드 수를 말한다. 호출되는 메소드가 많아질수록 객체의 복잡도는 증가한다.

RFC = |RS| RS는 클래스에 대한 반응 집합

$$RS = \{M\} \bigcup_{\forall i} \{Ri\}$$

$\{Ri\}$ = 메소드 i에 의해 호출되는 메소드 집합
$\{M\}$ = 클래스의 모든 메소드 집합

- 복잡도가 증가한다는 것은 유지보수가 어렵다는 뜻으로 RFC의 수치는 낮을수록 좋다.
- 만약 메소드의 많은 수가 메시지 전달로서 호출된다면, 객체에 대한 테스팅과 디버깅은 어렵게 된다.

3.2.8 McCabe의 사이클로메틱 복잡도

프로그램 구조의 복잡도를 측정하는 대표적인 메트릭스로서 절차적 소프트웨어의 복잡도 측정에 많은 도움을 주고 있다. 따라서 클래스의 복잡도가 중요시되는 객체지향 프로그램의 전체적인 복잡도를 나타내는 데는 부족하지만 메소드 구현의 복잡도를 나타내는 데는 사용할 수 있다. 이 메트릭스는 프로그램 수행경로를 프로그램 라인으로 표시하는 노드와 경로를 표시하는 간선으로 구성된 그래프를 나타낸 후 다음과 같은 공식을 적용한다. 복잡도(G) = 화살표의 수(A) − 노드의 수(N) +2.

각 노드는 원칙적으로는 프로그램 라인 하나를 의미하지만, 순차적으로 수행되는 여러 라인의 집합으로 생각하여도 좋다. 또한 소프트웨어가 여러 모듈들의 집합인 경우, 각 모듈들에 대한 복잡도만 평가하

여도 좋다[2]. 이 메트릭스의 결과는 다음과 같다.

- 복잡도가 5이하인 경우:매우 간단한 프로그램일 때
- 복잡도가 5-10인 경우:매우 구조적이며 안정된 프로그램
- 복잡도가 20이상인 경우:문제 자체가 매우 복잡하거나, 구조가 필요이상으로 복잡한 프로그램
- 복잡도가 50이상인 경우:매우 비구조적이며 불안정한 프로그램

복잡도와 품질의 관계는 복잡도 메트릭스가 소프트웨어 시험에서 오류가 있음을 예측할 수 있고, 유지보수의 용이성과도 관계가 있다. 이 모든 관계는 생산성과 직결됨을 알 수 있다. 하지만 이것은 결과만을 평가해줄 뿐 과정상의 도구는 되지 못하며, 구조의 복잡도에 치우친 나머지 데이타 복잡도는 무시하고 있다는 단점이 있다.

위의 메트릭스를 정리하면 다음 표와 같다.

〈표 1〉 메트릭간의 관계
〈Table 1〉 Relation among metrics

오브젝트 메트릭스	오브젝트 정의	오브젝트 속성	오브젝트간 통신
WMC	X	X	
DIT	X		
NOC	X		
RFC		X	X
CBO			X
LOCM		X	

다음은 로렌즈의 실험과 그 동안 객체 지향 메트릭스, 효율적인 클래스 설계 이론 등에 공통으로 나타나는 경험적인 측정과 메트릭스 규칙을 바탕으로 한 결과이다[4].

- Cyclomatic Number는 10 이하가 바람직하다.
- 클래스당 평균 멤버 함수의 수는 20개 이하가 좋다.
- 클래스당 평균 인스턴스 변수는 6개 이하가 좋다.
- 클래스의 상속 계층 구조는 6 이하가 바람직하다.
- 한 클래스의 CBO 값만 0이 아니고 다른 클래스의 CBO 값이 모두 0이면 객체 지향 설계라고 볼 수 없다.

위의 결과를 바탕으로 재사용 부품의 품질을 평가하기 위한 품질 목표를 다음과 같이 설정하였다.

〈표 2〉 메트릭의 한계치
〈Table 2〉 Limits of metrics

메트릭	한계치
NOC	1~7
WMC	1~20
DIT	1~6
NOM	5~7
CBO	1 이상
LCOM	0
RFC	0~6
Cyclomatic Number	1~10

3.3 소프트웨어 품질 특성과 품질 메트릭과의 관계

외부 품질 특성은 어떤 객관적인 메트릭으로 간단히 표현할 수가 없다. 여러 가지 내부 특성들이 복합적으로 작용하여 외부 품질 특성에 영향을 미치기 때문에 추정할 수밖에 없다. 예를 들어 소프트웨어의 유지보수용이도를 측정한다면 이해용이도, 변경용이도, 시험용이도가 관련되어 있다. 이들 외부 품질특성을 측정하려면 관련된 내부 특성, 예를 들면 제어흐름 복잡도와 같은 객관적 메트릭 방법을 결정하여 값을 구한 후 각 특성과의 관련함수에 적용하여 추정한

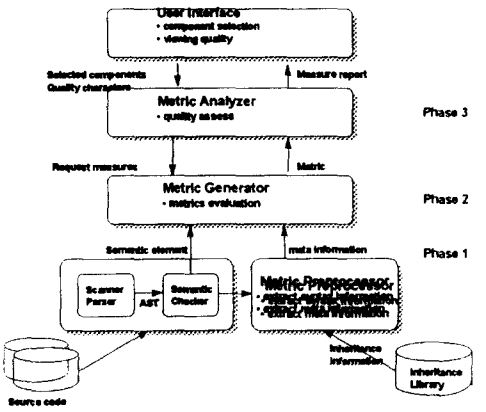
〈표 3〉 품질 특성과 메트릭과의 관계
〈Table 3〉 Relation of quality characteristics and metrics

	재사용	이해 용이	이식	유지보수	신뢰
NOC	X				
WMC	X				
DIT	X	X			
NOM		X			
CBO		X	X		
LCOM			X	X	
RFC			X	X	
Cyclomatic				X	X

다. 〈표 3〉은 재사용 부품의 외부 품질 특성을 측정하기 위하여 관련된 내부 메트릭들을 표시한 것이다.

4. 시스템 구조

품질평가 시스템은 (그림 4)와 같이 세 가지 단계로 구현된다. 첫 번째 단계인 품질평가를 위한 전처리단계에서는 재사용 부품 저장소에서 재사용부품을 읽어서 그 원시 코드의 구문적 구조를 분석하고 상속정보 저장소에서 상속정보를 읽어서 재사용 부품의 상속적인 구조를 파악하는 단계이다. 이 단계는 품질평가를 위한 기존 정보를 추출하는 역할을 수행한다. 두 번째 단계는 전 단계에서 추출된 품질평가를 위한 기본 정보를 이용해서 〈표 3〉의 첫 번째 칸에 나열한 여덟 가지 메트릭을 측정한다.



(그림 4) 품질 평가 도구의 시스템 구조
(Fig. 4) System architecture of quality assessment tool

4.1 입력 정보

- 재사용 부품의 원시코드(C++)
- 상속 정보

4.2 출력 정보

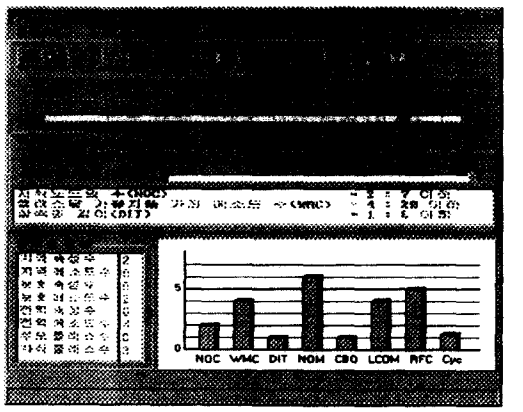
- 각 품질 특성
- 적합한 품질 특성 목록
- 부적합한 품질 특성 목록
- 종합적인 품질 평가 (품질 만족도)
- 한계치 목록

4.3 실행 화면

(그림 5)와 같이 상단부분에는 메뉴가 있고, 그 아래에는 각 품질기준에 따른 검증결과 표시되고, 아래 부분에는 한계치가 표시된다[10].

4.4 객체 모형

(그림 6)은 본 품질평가 시스템의 객체모형이다. 본 품질 평가시스템은 품질 서브시스템, 품질 분석 서브시스템, 사용자 인터페이스 서브시스템으로 구성된다.



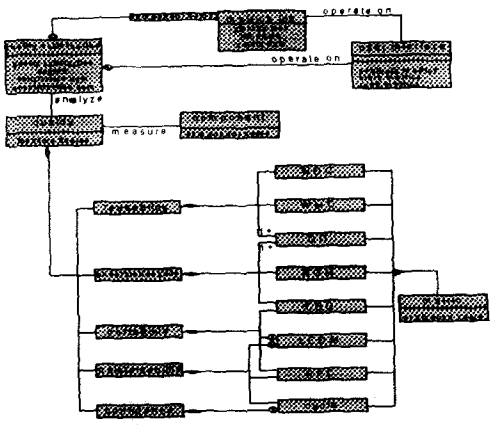
(그림 5) 사용자 인터페이스
(Fig. 5) User interface of quality assessment tool

4.4.1 품질 서브시스템

품질 서브시스템은 재사용 부품의 품질을 구조적 메트릭과 상속적 메트릭으로 나누어 평가하는 서브시스템이며 8가지 메트릭을 측정하기 위한 NOC 클래스, WMC 클래스, DIT 클래스, NOM 클래스, CBO 클래스, LCOM 클래스, RFC 클래스, cyclo 클래스, 모든 메트릭 클래스의 가상 클래스인 metrics 클래스들로 구성된다.

4.4.2 품질 분석 서브시스템

품질 분석 서브시스템은 평가한 메트릭을 본 시스템에서 주어진 한계치에 따라 평가, 분석하여 품질 만족도를 구하는 서브시스템이며 재사용성을 측정하기 위한 reusability 클래스, 이해용이성을 측정하기 위한 understandability 클래스, 이식성을 측정하기 위한 portability 클래스, 유지보수성을 측정하기 위한 ma-



(그림 6) 품질 평가 도구의 객체 모형
(Fig. 6) Object model of quality assessment tool

intainability 클래스, 신뢰성을 측정하기 위한 confidence 클래스, 모든 품질 클래스의 가상 클래스인 quality 클래스, 품질 만족도를 위한 quality satisfaction 클래스들로 구성된다.

4.4.3 사용자 인터페이스 서브시스템

사용자의 입력을 받고 평가한 품질을 수치적 방법과 도표로 출력하는 서브시스템이며 document 클래스, view 클래스, graph 클래스, server 클래스들로 구성된다.

5. 소프트웨어 품질평가 사례

5.1 품질평가 사례를 위한 예제 소스 프로그램

제시된 품질평가 기준에 대하여 실제 메트릭에 의하여 평가하는 과정을 보이기 위하여 부록에 있는 샘플 프로그램을 이용하였다. 샘플 프로그램은 간단한 레스토랑 관리를 위한 시스템의 재사용 부품이다.

5.2 자식 노드의 수(Node of Children)

현재의 클래스를 계승받은 클래스의 수이다.

5.3 클래스의 멤버함수중 가중치를 가진 멤버 함수의 수(Weighted Member per Class)

WMC 값은 멤버 함수의 가중치의 합인데 여기서 가중치는 그 멤버 함수의 사이클로매틱 수이다.

〈표 4〉 NOC의 계산 결과
〈Table 4〉 Metric results of NOC

클래스 이름	NOC 값
Worker	2
Waiter	1
Singer	1
SingingWaiter	0

〈표 5〉 WMC의 계산 결과
〈Table 5〉 Metric result of WMC

클래스 이름	WMC 값
Worker	4
Waiter	5
Singer	8
SingingWaiter	4

5.4 상속 트리의 깊이(Depth of Inheritance Tree)

상속 트리의 깊이는 각 클래스가 상속을 받은 깊이로 구한다.

〈표 6〉 DIT의 계산 결과
〈Table 6〉 Metric result of DIT

클래스 이름	DIT 값
Worker	1
Waiter	2
Singer	2
SingingWaiter	3

5.5 지역 메소드의 수(Number Of Methods)

〈표 7〉 NOM의 계산 결과
〈Table 7〉 Metric result of NOM

클래스 이름	NOM 값
Worker	5
Waiter	8
Singer	7
SingigWaiter	10

5.6 객체간의 결합도(Coupling Between Object)

4개의 클래스가 모두 입출력을 위하여 iostream 클래스와 결합하였으므로 CBO 메트릭스 값은 1이다.

〈표 8〉 CBO의 계산 결과
〈Table 8〉 Metric result of CBO

클래스 이름	CBO 값
Worker	1
Waiter	1
Singer	1
SingingWaiter	1

5.7 메소드 응집도의 결핍(Leak of COhesion in Method)

〈표 9〉 LCOM의 계산 결과
〈Table 9〉 Metric result of LCOM

클래스 이름	LCOM 값
Worker	4
Waiter	4
Singer	4
SingingWaiter	4

5.8 클래스에 대한 반응도(Responsibility For a Class)

〈표 10〉 RFC의 계산 결과
〈Table 10〉 Metric result of RFC

클래스 이름	RFC 값
Worker	5
Waiter	8
Singer	7
SingigWaiter	10

5.9 클래스의 멤버 함수의 사이클로매틱 수(Cyclomatic Number)

품질 평가 기준을 적용하여 클래스의 품질 만족도를 계산하면 다음과 같다. 먼저 선정된 8개의 품질 메트릭을 측정한 결과는 〈표 12〉와 같다.

〈표 11〉 복잡도의 계산 결과
〈Table 11〉 Metric result of complexity

클래스	멤버 함수	사이클로매틱 수	클래스의 복잡도
Worker	show	1	1.3
	data	1	
	get	2	
Waiter	set	1	1.2
	show	1	
	data	1	
	get	2	
Singer	set	1	2
	show	1	
	data	1	
	get	6	
SingingWaiter	set	1	1
	show	1	
	data	1	
	get	1	

〈표 12〉 품질 메트릭의 측정 결과
〈Table 12〉 Quality metrics results

	CN	WMC	DIT	WIT	NOC	CBO	RFC	LCOM
Worker	1.3	4	1	2	2	1	5	4
Waiter	1.2	5	2	1	1	1	8	4
Singer	2	8	2	1	1	1	7	4
Singing Waiter	1	4	3	1	0	1	10	4

위의 결과를 〈표 3〉에 따라 만족하는 항목의 수를 그 품질을 평가하기 위해 필요한 메트릭 수로 나누면 결과는 아래의 〈표 12〉와 같다.

〈표 13〉 품질 평가 결과
〈Table 13〉 Results of quality assessment

	Worker	Waiter	Singer	Singing Waiter
신뢰성	100	100	100	100
재사용성	100	100	100	100
유지 보수성	66.7	33.3	33.3	33.3
이식성	66.7	33.3	33.3	33.3
이해 용이성	100	100	100	100

6. 성능 비교분석

품질 평가 시스템은 크게 다음 세가지 측면에서 검토할 수 있다. 먼저 품질 측정의 대상이 되는 원시코드에 대한 사항이다. 품질 평가도구는 프로그래밍 언어로 표현된 내용을 충분히 구문분석하여 평가할 수 있어야 한다. REBOOT의 경우 최근 표준 C++ 언어를 다루고 있으나 QUALMS는 단순한 절차적 프로그래밍 언어, 그리고 ESCORT의 경우는 C++의 충분한 구문을 다루지 못하고 있다.

또 다른 비교 관점은 측정 메트릭의 다양성이다. REBOOT의 경우는 ISO에서 제시하는 일반적인 소프트웨어의 품질 요소보다는 호환성, 유연성, 이해용이성, 적합성 등 재사용 관점에서만 평가하고 있다. QUALMS는 prime flowgraph를 기초로 주로 소프트웨어의 구조에 관련되는 메트릭을 측정하고 있다. 또한 ESCORT는 소프트웨어 생명주기 전체를 커버하는 평가도구라는 점에서 장점을 가지고 있으나 품질 평가 관점이 유지보수에 치우친 면이 있다. 예를 들어 기능 사이즈, 이해용이성, 복잡성 등만을 이용하여 모듈을 종합 평가한다. 또한 객체지향에 관련된 메트릭도 응집력과 복잡도 두가지뿐이다. 반면에 본 연구에서 제안한 품질 평가 도구는 5 가지 측면의 품질 특성을 8 가지 메트릭으로 측정하여 판단한다. 또한 객체지향 프로그램의 여러 가지 구문을 충분히 인식하여 그 특성을 메트릭으로 측정할 수 있다.

마지막으로 도구의 환경적인 측면에서 비교해 보면 REBOOT는 OSE/MOTIF 상에서 서버 CASE 도구와 함께 운용된다. 따라서 클라이언트에서는 서버의 REBOOT 시스템으로 들어가야 하는 불편이 있다. QUALMS의 경우는 구체적인 도구 환경이 알려지지 않고 있으며 ESCORT는 역시 standalone 형식으로 운용되고 있다.

7. 결 론

소프트웨어 품질 평가는 그 목적, 대상 그리고 결과에 대한 객관적인 원칙이 설정되어야 하며 그 측정 방법과 과정에 대해서는 합당한 기준이 설정되어야 한다. 품질 측정 및 평가는 객관적인 측면과 아울러 주관적인 측면도 불가피하다. 이 주관적인 측면은 최대한의 객관성과 일관성이 유지되도록 표준화된 방법, 지침 및 기준을 사용해야 한다. 따라서 본 논문에서는 ISO/IEC의 품질기준을 객체지향적인 입장에서 재사용 부품의 품질기준을 마련하였고 분산 객체 환경인 CORBA환경에서 구현하기 위해서 품질 검증 시스템의 설계를 하였다.

본 연구에의하여 개발된 품질평가 도구의 특징은 객체지향 프로그램에 대한 객관적 메트릭 방법들을 이용하여 품질 특성, 예를 들면 신뢰성, 재사용성, 유지보수성 등을 판단할 수 있는 기초자료를 제공하고 있다. 대부분의 객체지향 메트릭 연구는 원시코드의 복잡도, 크기 등의 일면을 객관적으로 나타내는 데 그치고 있고 품질 수준의 판단에까지 연결시키지 못하고 있다. 또한 서버에 있는 재사용 라이브러리를 여러 클라이언트에 있는 재사용 브라우저를 이용 품질을 검사해 보고 재사용 여부를 판단할 수 있도록 분산 환경에 구현되었다. 다만 객관적 메트릭과 품질 요소들의 관계는 더 많은 사례연구와 개발 팀의 성격을 고려한 분석으로 모형화시킬 필요가 있다.

참 고 문 헌

[1] Gutorm Sonder and Reidar Conradi, "The REBOOT Approach to Software Reuse", The Journal of System and Software, 1995, pp. 56-69.

[2] McCabe T. L, "A Complexity Measure", IEEE Trans. on SE, Vol. SE-2, 1976, pp. 308-320.

[3] Robert C. Sharble and Sammuel S. Cohen, "The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods", ACM SIGSOFT SE Note, Vol. 18, No. 2, 1993, pp. 60-73.

[4] 한규정, "객체 지향 개발에서의 메트릭스 적용", 소프트웨어공학회지 1994년 6월, pp. 107~112.

[5] T. Biggerstaff, A. J. Perlis, Software Reusability, Vol. 1: Concepts and Models, Addison-Wesley, 1989.

[6] P. A. V. Hall, Overview of Reverse Engineering and Reuse Research, Information and Software Technology 34, pp. 239-249, 1992.

[7] C. W. Krueger, Software Reuse, ACM Comput-

ing Surveys, Vol. 24, pp. 131-183, 1992.

[8] E. Karlsson, Sftware Reuse : a holistic approach, John Wiley & Sons, REBOOT Methodology Handbook, 1995.

[9] 임기욱 외, RODEO : OMA 기반 객체지향 멀티미디어 소프트웨어 개발환경 연구, 정보통신부, 1996.

[10] 최은만 외, 객체부품화 및 품질평가 기술 개발, 한국전자통신연구소, 1996.

[11] G. Caldiera, V. Basili, Identifying and Qualifying Reusable Software Components, IEEE Computer, pp. 61-70, 1991.

[12] 양해술 외, 소프트웨어 품질 평가 도구 ESCORT의 구현, I, II, 한국정보과학회 봄 학술발표논문집, pp. 661-668, 1994.

부록: 품질 평가 사례를 위한 원시코드

```
// workermi.h  -- working classes
#include "strng2.h"

class Worker {
private:
      String fullname;
      long id;
protected:
      virtual void data() const;
      virtual void get();
public:
      Worker() : fullname("no one"), id(0L) {}
      Worker(const String & s, long n) : fullname(s), id(n) {}
      virtual void set();
      virtual void show() const;
};

class Waiter : virtual public Worker {
private:
      int panache;
protected:
      void data() const;
      void get();
public:
      Waiter() : Worker(), panache(0) {}
      Waiter(const String & s, long n, int p=0) :
            Worker(s, n), panache(p) {}
      Waiter(const Worker & wk, int p = 0) :
            Worker(wk), panache(p) {}
      void set();
      void show() const;
};

class Singer : virtual public Worker {
protected:
      enum {other, alto, contralto, soprano,bass, baritone,tenor};
      enum {Vtypes = 7};
      void data() const;
      void get();
private:
      static char *pv[Vtypes];          // string equivs of voice types
      int voice;
public:
      Singer() : Worker(), voice(other) {}
      Singer(const String &s, long n, int v=other) :
            Worker(s, n), voice(v) {}
      Singer(const Worker & wk, int v = other) :
            Worker(wk), voice(v) {}
      void set();
      void show() const;
};

class SingingWaiter : public Singer, public Waiter {
protected:
      void data() const;
      void get();
public:
      SingingWaiter(){}
      SingingWaiter(const String &s, long n, int p=0, int
v=Singer::other)
            : Worker(s,n), Waiter(s, n, p), Singer(s, n, v) {}
      SingingWaiter(const Worker & wk, int p = 0, int v =
Singer::other)
            : Worker(wk), Waiter(wk,p), Singer(wk,v) {}
      SingingWaiter(const Waiter &wt, int v = other)
            : Worker(wt),Waiter(wt),inger(wt,v) {}
      SingingWaiter(const Singer & wt, int p = 0)
            : Worker(wt),Waiter(wt,p),inger(wt) {}
      void set();
      void show() const;
};

#include "workermi.h"     // workermi.cpp -- working class methods
#include <iostream.h>

void Worker::set() { cout << "Enter Worker's name: "; get();  }
void Worker::show() const {cout << "Category: Worker\n"; data(); }
void Worker::data() const {
      cout << "Name: " << fullname << "\n";
      cout << "Employee ID: " << id << "\n";
}
void Worker::get() {
      cin >> fullname; cout << "Enter Worker's ID: ";
      cin >> id;
      while (cin.get() != '\n') continue;
}
void Waiter::set() {
      cout << "Enter waiter's name: ";
      Worker::get();
      get();
}
void Waiter::show() const {
      cout << "Category: waiter\n";
      Worker::data();
      data();
}
void Waiter::data() const {
      cout << "Panache rating: " << panache << "\n"; }
void Waiter::get() {
      cout << "Enter waiter's panache rating: ";
      cin >> panache;
      while (cin.get() != '\n') continue;
}
char *Singer::pv[Singer::Vtypes]={"etc","al","me","so","ba","bari","te"};
void Singer::set() {
      cout << "Enter Singer's name: ";
      Worker::get();
      get();
}
void Singer::show() const {
      cout << "Category: Singer\n";
      Worker::data();
      data();
}
void Singer::data() const {cout << "Vocal range:" << pv[voice] << "\n";
}
void Singer::get() {
      cout << "Enter number for Singer's vocal range:\n";
      for (int i = 0; i < Vtypes; I++) {
            cout << i << ": " << pv[i] << "    ";
```

```
            if ( i % 4 == 3) cout << '\n';
        }
        if (i % 4 != 0) cout << '\n';
        cin >>  voice;
        while (cin.get() != '\n') continue;
}
void SingingWaiter::data() const { Singer::data();     Waiter::data(); }
void SingingWaiter::get() { Waiter::get();             Singer::get(); }
void SingingWaiter::set() {
        cout << "Enter singing waiter's name: ";
        Worker::get();          get();
}

void SingingWaiter::show() const        {
        cout << "Category: singing waiter\n"; Worker::data(); data();
}
```

최 은 만

1982년 동국대학교 전산학과 졸업(학사)
1985년 한국과학기술원 전산학과 졸업(석사)
1993년 미국 일리노이 공대 전산학과 졸업(박사)
1985년~1988년 한국표준연구소 연구원
1988년~1989년 데이콤 주임연구원
1993년~현재 동국대학교 정보산업학부 조교수
관심분야: 객체지향 소프트웨어 공학, 소프트웨어 재사용, 역공학, 유지보수

남 윤 석

1984년 아주대학교 산업공학과 (학사)
1989년 Polytechnic Univ.(New York), Dept. of Industrial Engineering(석사)
1992년 Polytechnic Univ.(New York), Dept. of Industrial Engineering(박사)
1993년~현재 한국전자통신연구원 정보공학연구실 선임연구원
관심분야: 소프트웨어 공학, 객체지향 재사용 기술 등