Java 프로그램의 품질평가를 지원하는 메트릭 측정 시스템

(Metrics Measurement System Supporting Quality Evaluation of Java Program)

박 옥 자 † 유 철 중 †† 장 옥 배 ††
(Ok-Cha Park) (Cheol-Jung Yoo) (Ok-bae Chang)

요 약 최근 가장 대표적인 객체지향 언어로 사용되는 Java는 일반적인 애플리케이션 뿐만 아니라 인터넷/인트라넷 기반 프로그램 개발, 나아가 컴포넌트 기반 개발에 이르기까지 다양한 분야에서 개발 언어로 사용하고 있다. 따라서 개발된 프로그램의 재사용 및 유지보수 관점에서 프로그램 품질평가는 보다 중요한 쟁점이 되고 있으므로 기존의 Java 애플리케이션을 포함하여 현재 개발된 프로그램의 품질평가에 필요한 메트릭 측정이 필요하다. 하지만, 이미 제안된 객체지향 소프트웨어 메트릭이 현재의 Java 프로그램의 특성에 적합한지에 대한 타당성 검증이 필요하므로 본 논문에서는 기존의 객체지향 메트릭이 Java 프로그램에 적합한지 여부를 결정하기 위해 필요한 메트릭 측정 시스템을 구축하여 Java 프로그램에 적합한 메트릭 제안을 지원하고자 한다.

본 시스템은 Briand가 기존의 객체지향 소프트웨어 메트릭을 수학적으로 정형화시켜 분류한 메트릭을 Java 프로그램에 적용시켜 제안된 메트릭이 프로그램에 타당성 있는지 검증함으로써 명확한 품질평가도구 개발을 지원하고자 한다. 본 시스템을 통해 Java 소스 프로그램으로부터 정량적 정보를 보다 빠르고 정확하게 산출함으로써 기존의 객체지향 메트릭에 대한 검증을 비교 및 분석 수행할 수 있으며, 타당성 문제가 있다면 새로운 메트릭의 제안 및 보완을 고려함으로써 Java 프로그램에 적합한 메트릭 확립을 가능하게 할 것이다.

Abstract Java, used as the most representative object-oriented language, is" becoming the popular language for Internet/Intranet based program development. Moreover, it is used for development language in a variety of areas such as component based development language. In the view of reuse and maintenance of developed program, quality evaluation of program is becoming a more important issue. So, metrics measurement for quality evaluation of program that is developed at present including existing Java application is necessary. However, it is necessary that whether existing object-oriented software metrics is suitable on Java program is to be validated So, in this paper, we build an automated metrics measurement system that needs to validate on object-oriented software metrics and wish to support metrics measurement that is to determine it.

The purpose of this system is to support a precise quality evaluation tool. In this system, we apply the metrics classified by Briand. Briand classified the metrics by formalizing mathematically them to verify feasibility of existing object-oriented software metrics. Using the proposed system, we can make comparison and analysis of validation on existing object-oriented metrics by calculating quantitative information more rapidly from Java source program. If there is any problem in feasibility of the metrics, we can establish a suitable metrics on Java program by considering reinforcement of the existing metrics or proposing new metrics.

† 학생회원 : 전북대학교 전산통계학과
ojpark@cs.chonbuk.ac.kr
†† 종신회원 : 전북대학교 컴퓨터과학과 교수
전북대학교 영상·정보신기술연구소 연구원
cjyoo@moak.chonbuk.ac.kr
okjang@moak.chonbuk.ac.kr

논문접수 : 2000년 7월 14일
심사완료 : 2001년 3월 3일

1. 서 론

최근 가장 대표적인 객체지향 언어로 사용되는 Java는 소프트웨어공학 관점에서 수정이 용이하고 적은 비용으로도 유지보수가 가능하며 재사용성과 확장성을 가지게 하는 언어이다[1]. 따라서 일반적인 애플리케이션에서 인터넷/인트라넷 기반 프로그램 개발에 이르기까지 다양한 분야에서 개발 언어로 사용하고 있고 최근에는 컴포넌트 개발 방법론에 가장 적합한 언어로 인식되면서 프로그램 품질평가에 대한 인식이 보다 강조되고 있다[2]. 또한, 기존에 구축된 Java 애플리케이션을 컴포넌트화(componentization)할 필요성이 대두되면서 프로그램 선정기준에 필요한 품질평가는 중요한 쟁점이 되고 있다. 따라서, 기존의 Java 애플리케이션을 포함하여 현재 개발된 소스 프로그램에 대한 품질평가는 반드시 필요하다.

하지만, 기존에 제안된 객체지향 소프트웨어 메트릭[13, 29, 30]의 대부분은 분석자들의 경험적 연구만을 바탕으로 이루어졌기 때문에 같은 평가 기준에서도 서로 다른 메트릭(metrics)이 적용되거나 사용자의 관점에 따라 메트릭 적용 범위가 달라지는 등 일관성이 없다. 이는 소프트웨어 품질과 메트릭과의 관계가 모호하고 소프트웨어 품질 속성이 만족되기 위해서 어떤 메트릭 집합이 만족되어야 하는지, 반대로 메트릭이 만족되었을 경우 품질의 어떤 측면이 개선되는지 등에 대한 이론적 근거가 제대로 제공되지 않고 있기 때문이다[28]. 따라서, 소프트웨어 품질평가 적용에 있어서 명확하게 정의된 평가 기준과 메트릭 집합을 수학적으로 정형화시켜 이론적 근거를 기반으로 제공되는 메트릭의 측정이 필요하다[3]. 비록 Ejiogu[31]가 정의한 효과적인 소프트웨어 메트릭이 포함해야 하는 6가지 요건과 같이 일정한 범주를 정해서 객체지향 메트릭 적용의 필요성에 대해 이론적 근거를 제시한 경우도 있지만 마찬가지로 수학적으로 정형화되어 있지 않아 메트릭 적용에 있어서 모호성을 유발한다.

본 논문에서는 이와 같은 필요성에 근거하여 먼저 기존의 객체지향 메트릭을 통합하여 수학적으로 정형화시킨 Briand의 속성기반 메트릭을 Java 소스 프로그램에 적용시켜 정량적 정보를 얻도록 구축하였다. 또한, 프로그램 이해 관점에서 Java 프로그램의 특성상 얻기 어려운 정보 즉, 상속(inheritance), 메소드 호출관계(method invocation), 그리고 객체 생성과 같은 소스코드 이면의 정보들을 보다 정확하게 얻기 위하여 Java 바이트코드(bytecode)를 분석하여 소스 프로그램으로부터 정량적으로 얻어진 메트릭의 정보 및 이해를 지원하였다.

본 측정 시스템은 다음과 같이 크게 세 가지 모듈로 구성된다. 첫째, 소스 모듈은 측정하고자 하는 Java 소스 프로그램을 보여준다. 이것은 본 도구가 최종 품질평가 정보를 얻기 보다는 기존의 메트릭이 Java 프로그램에 어떻게 적용되는지 검증하고자 하는데 목적이 있기 때문에 타당성 비교 분석을 위해 필요하다. 둘째, 프로그램의 구조적 이해를 도울 수 있는 프로그램 이해지원 모듈이다. 메트릭의 타당성 검증을 위해서는 메트릭 측정 뿐만 아니라 프로그램 이해도 중요하므로 본 도구는 Java 소스코드로부터 얻기 어려운 정보를 바이트코드로부터 추출하여 이를 직관적으로 이해할 수 있도록 비주얼(visual)하게 제공하였다. 마지막으로, 품질평가 모듈로서 측정 기준은 Briand의 속성기반 메트릭을 이용하여 기존의 메트릭을 수학적으로 정형화하여 분류시킨 크기(Size), 길이(Length), 복잡도(Complexity), 결합도(Coupling), 응집도(Cohesion) 속성을 만족시키는 객체지향 메트릭을 적용하였다[4, 5].

본 논문의 구성은 다음과 같다. 2장에서 기존에 제안된 객체지향 소프트웨어 메트릭과 그에 대한 문제점을 알아보고 나아가 가장 잘 알려진 객체지향 소프트웨어 품질평가 시스템에 대해 알아본다. 3장에서는 본 시스템에서 적용한 평가 기준인 Briand의 객체지향 소프트웨어 메트릭의 이론적 근거에 대해 간단히 살펴보고, 프로그램 이해 및 분석 모듈의 근거가 되는 Java 바이트코드에 대해 알아본다. 4장에서는 시스템의 설계 및 구현에 대한 내용으로서 시스템 구조와 인터페이스(user interface) 등에 대해 서술하였으며, 5장에서는 실행 결과 및 비교·분석을, 그리고 마지막 6장에서는 결론 및 향후연구 방향을 제시한다.

2. 객체지향 소프트웨어 메트릭

소프트웨어 유지보수에 대한 중요성이 증가하면서 소프트웨어에 대한 품질평가 도구와 프로그램 이해는 보다 중요하다. 소프트웨어 공학 관점에서 품질평가는 설계 단계부터 최종 구현 단계에 이르기까지 각 단계별 메트릭을 적용함으로써 최종적으로 프로그램 인수 후에도 수정 및 보완이 필요 없는 프로그램이 되도록 개발 방법론을 지향하고 있다[6].

하지만, 현재까지 개발된 대부분의 시스템은 최종 산출물(product)인 소스 프로그램을 기반으로 한 품질평가에 의존하고 있다. 이는 아직까지 단계별 품질평가를 지원할 수 있는 명확한 메트릭이 제시되지 않아 설계 단

계의 품질평가에 대한 신뢰성이 낮은데다가 직접적인 프로그램 평가 및 유지보수는 대부분 소스수준의 산출물에서 이루어지고 있기 때문에다. 따라서 소스수준의 품질평가는 중요하고 이를 바탕으로 하는 자동화된 품질평가는 필요하다.

소프트웨어공학회에서 정의한 것에 따르면 메트릭은 소스코드 및 설계의 구조적 품질을 나타내는 지표이다[4, 6, 7]. 따라서, 소프트웨어 품질평가 메트릭은 메트릭 적용 범위, 메트릭이 적용될 수 있는 관점의 이해, 메트릭 정의와 해석에 대한 체계, 메트릭들이 근거를 두고 있는 기준과 모델의 제안, 그리고 측정의 자동화 및 응용과 같은 분야에서 지속적으로 연구가 이루어 졌다[8, 9]. 본 장에서는 기존에 제안된 객체지향 소프트웨어 메트릭을 간략히 살펴보고 그에 대한 문제점을 분석해 보기로 한다.

먼저 Lorenz[10]은 절차지향 개발 방법론에서 객체지향개발 방법론으로 전환해 오면서 소프트웨어 개발에 대한 품질평가 기준을 프로세스(Process), 산출물(Product), 자원(Resources)의 관점에서 OOMetrics (Object-Oriented Metrics)를 제안하였다. 각 메트릭 단위는 보다 구체적으로 상세 분리되는데 특히, 산출물 메트릭은 크기(Size), 구조(Structure), 품질(Quality) 단위로 이루어졌다[11]. 이와 같은 산출물은 객체지향 프로그램의 특성에 맞게 클래스와 메소드 단위로 상세 분류하여 있도록 표 1과 같이 제안하였다.

한편 Chidamber와 Kemerer(이하 CK)[12, 13]는 객체지향 소프트웨어 품질평가를 1) 클래스 당 가중치를 갖는 메소드 수(Weighted Methods Per Class : WMC), 2) 상속 트리의 깊이(Depth of Inheritance : DIT), 3) 자노드의 수(Number Of Children : NOC), 4) 객체간의 결합도(Coupling Between Object classes : CBO), 6) 클래스에 대한 반응도(Response For a Class : RFC), 7) 주어진 인스턴스(*instance*)변수를 참조하는 클래스 내의 다른 메소드 수 (Lack Of COhesion in Methods : LCOM)를 기준으로 제시하였다.

Herrison[14]은 Lorenz나 CK와는 다르게 객체지향의 가장 대표적인 특성인 1) 캡슐화(Encapsulation), 2) 상속(Inheritance), 3) 결합도(Coupling), 4) 다형성(Polymorphism)을 기준으로 메소드와 애트리뷰트(attribute) 범위에 해당하는 메트릭을 제안하였다.

위와 같이 Lorenz는 전체적인 시스템 평가가 가능하도록 메트릭을 제안한 반면 Harrison은 CK의 시스템 레벨의 측정 기준에 상호보완이 될 수 있는 객체지향 특성을 기준으로 메트릭을 제안하였다. 이 세 분야의 메트릭은 가장 대표적인 객체지향 소프트웨어 메트릭으로서 많은 논문들에서 참조되고 있고 실무에서 적용되고 있다. 하지만, 다음과 같이 몇 가지의 문제점을 안고 있어 그에 대한 타당성 여부도 계속 논의되고 있다.

표 1 OOMetrics의 상세분류

측정 단위	분류	메트릭	특징
메소드	크기	메시지 전달 수 라인 수	메소드 크기 측정
	내부	복잡도(McCabe 기준) 메시지 전달 스트링 수	메소드내의 복잡도
	외부	메소드 당 파라미터 수 커맨드 메소드 비율	메소드간의 품질평가
	상속	메소드 다중상속 메소드 상속	메소드 상속에 관한 측정
클래스	크기	인스턴스 메소드 수 인스턴스 변수 수 클래스 메소드 수 클래스 변수 수	클래스 크기
	내부	클래스 응집도 인스턴스 변수 사용빈도수	클래스내의 복잡도, 응집도
	외부	클래스 결합도 클래스 재사용	클래스간의 품질평가
	상속	계층적 중첩 레벨 다중 상속	클래스 상속에 관한 측정

프로그램 품질평가의 가장 중요한 쟁점은 수학적으로 검증된 수식과 이를 토대로 시행된 경험적 연구에서 얻어진 표준화된 메트릭이다. 또한, 신뢰성, 유지보수성, 테스팅과 같은 시스템의 외부 품질 속성간의 인과관계까지 포함시켜 모든 프로그램에서도 일관성 있는 결과를 보여줄 수 있도록 일반화시켜야 하는데 지금까지 제안된 메트릭 대부분은 위와 같이 단지 경험적 연구만을 토대로 얻어진 결과에 의지하였기 때문에 신뢰성이 부족하다[3, 14, 16]. 또한 기존에 제안된 메트릭을 프로그램에 적용시키는 경험적 검증과정에서도 그 결과가 타당하지 않다고 고려되면 분석자들은 기존의 메트릭을 임의적으로 수정하거나 새롭게 제안하였다. 하지만 마찬가지로 분석자들이 경험적 연구를 통해서 결론을 얻어내는 실제 과정은 단지 본인이 제안한 메트릭이 측정하려는 목적에 타당한지 여부만을 검사하게 되는 위험성을 낳게 되므로 각 메트릭이 경험적으로 타당하다고 할지라도 일반화 또는 표준화시키기 어렵게 되므로 결과적으로 같은 문제점을 낳게 된다[3].

이와 같은 문제점의 근본적인 원인은 우선적으로 품

질평가에 필요한 평가 기준이 명확하기 않고 분석자의 관점에 따라 같은 평가 기준을 갖고도 서로 다른 개념으로 접근하여 사용하고 있기 때문이다[3, 5, 28]. 예를 들어 CK는 WMC를 프로그램의 크기에 적용되는 메트릭로 생각한 반면 Briand는 CK가 결합도로 제안한 RFC도 크기(Size) 메트릭에 포함된다고 정의하였다. 이와 같은 관점의 불일치는 사용자 입장에서는 적용기준에 많은 혼란을 가져올 수 있고 보다 포괄적인 품질평가 기준에 있어서 선택의 어려움을 가져온다. 또한, 실제 명확한 개념이 정립되지 않는데서 출발하는 품질평가는 서로 다른 소프트웨어 특징을 추출하게 되는데 그것들이 명확히 분리될 때까지, 그리고 그들의 유사성과 차이점이 명확히 연구될 때까지 각각의 개념을 소프트웨어 메트릭 정의와 관련지어 어떤 일치되는 속성에 도달하는 것은 불가능하게 된다. 따라서, 품질평가에 필요한 메트릭을 적용함에 있어서 가장 중요한 것은 논쟁의 원칙이 되는 일관성 있는 개념 정의가 무엇보다도 중요하다[3, 28].

본 도구에서는 위와 같은 문제점을 해결하기 위해서 3장에서 설명할 Briand의 속성을 만족하는 메트릭을 기반으로 하여 시스템에 적용하고자 한다.

3. Java 프로그램의 품질평가 지원도구의 설계

이 장에서는 2장에서의 제시한 문제점을 보완하고, 구축한 시스템의 이론적 근거가 되는 Briand의 속성기반 메트릭과 이해 모듈 추출에 있어서 기반이 된 Java 바이트코드에 대해 살펴보기로 한다.

3.1 Briand의 속성 기반 소프트웨어 메트릭

Briand는 객체지향 소프트웨어 메트릭의 모호성을 해결하기 위해서는 먼저 품질평가에 있어서 메트릭이 실제 의미하는 것이 무엇인지 명확하게 규정하는 것이 가장 중요하다고 보고 소프트웨어공학회에서 표준화하려는 품질평가 측정 개념을 선정하여 정의하였다[5]. 또한, 이 개념을 상세히 분류하여 각 측정개념에 대해 만족할 수 있는 속성 집합을 수학적으로 정형화하여 기존에 제안된 다양한 객체지향 메트릭을 이 속성집합에 적용시킴으로써 이 속성을 만족하는 메트릭만을 추출하여 분류하였다.

표 2는 Briand가 정의한 측정 개념과 그에 대한 간략한 정의이다.

Briand의 메트릭 정의 방법은 다음과 같다. Briand는 메트릭을 정의하기 이전에 객체지향 속성에 대한 기본적 정의를 수학적으로 나타내는데 먼저 시스템과 모듈

표 2 Briand가 정의한 품질평가 측정 개념

측정 개념	정 의
크기	시스템의 크기와 관련된 메트릭으로서 사용자에게 가장 직관적인 값을 전달한다.
길이	크기와 마찬가지로 크기와 관련된 개념이지만 크기가 물리적으로 볼륨(volume)과 중요도(weight)와 같은 개념이라면 길이는 1차원적인 개념으로 정의하였다. Briand는 객체지향 소프트웨어 특성에서 상속개념을 구별해주기 위해 크기와 구별되는 길이 속성을 정의하였다.
복잡도	소프트웨어 품질평가에서 시스템의 특성과 관련된 메트릭으로서 소프트웨어 시스템의 신뢰성과 유지보수성에 관한 중요한 정보를 예측하는데 사용한다. Briand는 소스코드로부터의 심리적 복잡도를 제외하고 본질적 속성만을 정의하였다.
응집도	모듈이나 모듈화 시스템과 관련된 메트릭으로서 객체지향 특성 중 캡슐화를 측정하기도 한다.
결합도	응집도과 마찬가지로 모듈이나 모듈화 시스템과 관련된 메트릭으로서 시스템에서 다른 모듈에 속해있는 요소간의 관계성(relationship) 정도를 측정한다.

을 표현함에 있어서 시스템 S는 $<E, R>$로 표현되고, E는 S의 요소(elements)이고 R은 S의 요소들간의 관계를 나타내는 쌍($R \subseteq E \times E$)의 관계로 표현된다. 또한 모듈간의 관계를 포함(inclusion), 결합(union), 교차(intersection), 분리(disjoint)로 정의하였다. 각 모듈간의 관계는 다음과 같다.

Inclusion. 만약 $m_i \subseteq m_j$이고 $R_{mi} \subseteq R_{mj}$이면, 모듈 $m_i = <E_m, R_m>$은 $m_j = <E_{mj}, R_{mj}>$이다.

Union. $m_i=<E_{mi}, R_{mi}>$와 $m_j=<E_{mj}, R_{mj}>$의 결합은 모듈 $<E_{mi} \cup E_{mj}, R_{mi} \cup R_{mj}, R_{mi} \cup R_{mj}>$이다.

Intersection. $m_i=<E_{mi}, R_{mi}>$ 그리고 $m_j=<E_{mi}, R_{mj}>$의 교차는 $<E_{mi} \cap E_{mj}, R_{mi} \cap R_{mj}>$이다.

Empty module. 모듈 $<\varnothing, \varnothing>$은 empty 모듈이다.

Disjoint module. 만약 $m_i \cap m_j = \varnothing$이면, 모듈 m_i, m_j는 분리 모듈이다.

두 번째로 모듈화 시스템(Modular System)을 정의함에 있어서 모듈러 시스템 S는 $<E, R, M>$으로 표현되고 S는 정의 1에 따라 $<E, R>$, 그리고 M은 다음과 같이 S의 모듈의 집합으로 나타내었다.

$\forall e \in E(\exists m \in (m=<E_m, R_m> \text{ and } e \in E_m))$ and
$\forall m_i, m_j \in M(m_i=<E_{mi}, R_{mi}> \text{ and } m_j=<E_{mj}, R_{mj}>$
and $E_{mi} \cap E_{mj} = \varnothing$

이를 바탕으로 Briand는 그가 정의한 다섯 가지 품질평가 측정 개념을 토대로 각 측정개념에 필요한 속성을 수학적으로 정의하였다. 예를 들어 크기(Size) 메트릭은 다음과 같이 세 가지 속성을 정의하였다.

속성 1 : Nonnnegativity

$Size(S) \geq 0$

한 시스템이 존재하는 한 시스템의 크기는 언제나 0보다 크다.

속성 2 : Null Value

$E=\varnothing \Rightarrow Size(S)=0$

시스템을 구성하는 요소가 없으면 시스템의 크기는 0이다.

속성 3 : Module Additivity

$(m_1 \subset S \text{ and } m_2 \subset S \text{ and } E=E_{m1} \cup E_{m2} \text{ and } E_{m1} \cap E_{m2}=\varnothing)$
$\Rightarrow Size(S) = Size(m_1) + Size(m_2)$

S의 요소 E가 m_1, m_2로 구성될 경우 공유하는 정보가 없다면 두 모듈 m_1, m_2의 합과 같다는 것을 의미한다. 이것은 다시 모듈이 병합되어 구축될 경우 공통 요소들 때문에 시스템의 크기가 각 모듈들의 합보다 클 수가 없다는 것을 얻어낼 수 있다.

같은 방법으로 Briand는 길이, 복잡도, 응집도, 결합도에 대한 속성을 정의하였다. 지금까지 크기, 길이, 복잡도, 결합도, 응집도를 기준으로 한 Briand의 속성 기반 메트릭이 수학적 개념을 바탕으로 하기 때문에 명확한 품질평가의 기준이 될 수 있고 경험적 연구의 바탕이 될 수 있다.

표 2는 기존에 제안된 다양한 메트릭을 각 품질속성에 적용시켜 그 수학적 검증에 타당한 메트릭만을 추출하여 평가 기준별로 분류한 결과이다.

이로부터 얻어진 Briand의 속성기반 메트릭은 소프트웨어공학회에서 아직 명확히 표준화되어 있지 않은 품질평가 메트릭의 이론적 기반을 마련하는데 충분한 근거로 적용될 수 있고, 연구 및 분석가들에게는 좀더 표준화된 품질평가 가이드라인을 제공할 수 있으며 실사용자에게는 명확한 품질평가 방법들을 유도함으로써 일관성 있는 메트릭 측정이 가능하게 된다. 한편, 이와 같이 분류된 메트릭을 Java 프로그램에 적용시켜 특성이 타당하지 않거나 보완이 필요할 경우라도 Briand는 각 품질평가 기준에 대해 수학적으로 정형화한 이론적 근거를 제공하고 있으므로 이를 바탕으로 수정하거나 새로운 메트릭을 제안하는데 충분한 근거가 될 수 있다. 따라서, 본 시스템에서는 표 3에서 보여준 Briand의 속성을 만족하는 메트릭을 이용하였고, 이를 기반으로 한 Java 프로그램에 분석 결과를 토대로 그 메트릭에 대한 타당성과 문제점을 유도하게 된다.

표 3 Briand의 속성을 만족하는 메트릭

품질평가 기준	메트릭	참조
크 기	LOC : 라인 수	
	WMC : 클래스 당 가중치를 갖는 메소드 수	[12]
	NOC : 현재 클래스에 종속되어 있는 클래스 수	[12]
	RFC : 한 클래스가 받는 메시지 응답	[12]
길 이	DIT : 상속 트리의 깊이	[13]
복잡도	DF : 데이타 흐름	[17]
결합도	MPC : 메시지 전달 결합도	[18]
	RFC : 한 클래스가 받는 메시지 응답	[13]
	CBO : 클래스들간의 결합도	[13]
	CBO′ : 클래스들간의 결합도(상속 제외)	[12]
응집도	Coh : LCOM의 변형	[4] [19]
	TCC : 강한 클래스 응집도	[7]
	LCC : 약한 클래스 응집도	[7]

3.2 Java 바이트코드를 이용한 프로그램 이해 모듈

본 도구는 Java 프로그램의 품질평가 메트릭 결과를 얻기보다는 기존의 객체지향 메트릭이 Java 프로그램에 적합한지 여부를 검증하기 위한 전처리도구이므로 기존의 메트릭에 적용함으로써 얻어진 정량적 결과와 현재의 소스 프로그램을 비교 분석하면서 검증할 필요성이 있다. 따라서, Java 프로그램 특성상 얻기 어려운 정보 즉, 상속, 메소드 호출관계, 그리고 객체 생성과 같은 소스코드 이면의 정보들을 보다 정확하게 얻기 위하여 Java 바이트코드를 분석하여 메트릭 측정 및 이해에 적용하였다.

Java 바이트코드는 Java 소스 프로그램을 컴파일 한 결과물로서 보통 Java 클래스 파일로 불려진다. Java 소스프로그램을 컴파일하게 되면 .class 파일이 생성된다. 이 파일은 Java 소스코드 내에 정의된 클래스, 메소드, 변수들에 대한 정보를 포함하고 있으며, 컴파일 단계를 거친 일종의 중간코드 형태로서 다양한 플랫폼마다 동일하게 적용된다[20, 21].

클래스 파일의 구조는 다음과 같다.

표 4 Java 클래스 파일의 구조

```
class_file {
    u4              magic;
    u2              minor_number;
    u2              major_number;
    u2              constant_pool_count;
    cp_info         constant_pool
                    [constant_pool_count-1];
    u2              access_flags;
    u2              this_class;
    u2              super_class;
    u2              interfacecount;
    u2              interface[interface_count];
    u2              fields_counts;
    field_info fields[fields_count];
    u2              methods_count;
    method_info     methods[methods_count];
    u2              attributes_count;
    attribute_info  attributes[attributes_count];
}
```

클래스 파일은 위의 표 4와 같은 구조로 이루어져 있으며 여기서 u1, u2, u4는 각각 unsigned 1, 2, 4 바이트 값을 의미한다.

모든 클래스 파일의 첫 4바이트는 0xCAFEBABE고 두 번째와 세 번째는 자바 컴파일러의 주 버전, 부 버전 나타낸다. 자바 소스코드에서는 모든 클래스와 스트링, 정수 등은 constant_pool로 취급된다. constant_pool은 데이타의 길이가 일정하지 않은 항목들의 연속으로 되어 있으며, constant_pool[0]은 사용하지 않는다. 따라서 constant_pool[1]부터 constant_pool[constant_ pool _count-1]까지의 항목이 클래스 파일에 연속적으로 배치되어 있다. access_flags에는 변경자(modifier)의 종류가 저장되며, 자신의 수퍼(super) 클래스, 그리고 자기 자신을 나타내는 this 클래스와 this 클래스가 구현한 인터페이스, 필드, 메소드등이 저장된다. 마지막 attribute_field에는 클래스 파일 이름이 저장된다[21].

이 정보는 비록 바이트코드로 저장되어 있지만 이 코드를 역어셈블(disassemble)하면 클래스와 메소드의 제어흐름을 분석할 수 있도록 정보를 갖고 있다. 따라서, 소스코드에서 얻기 어려운 생성자를 비롯한 제어흐름에 영향을 주는 객체의 메소드들을 모아 분석하게 되면 클래스와 메소드, 또는 메소드와 메소드 사이의 내포 관계를 명확히 구분할 수도 있게된다.

JDK는 위와 같은 바이트코드의 정보를 쉽게 이해할 수 있도록 역어셈블러인 javap를 함께 지원한다. javap는 Java 클래스 파일을 쉽게 이해할 수 있도록 Op 코드(Opcode) 형태로 정보를 표현한다. 본 논문에서는 javap가 제공하는 Op 코드 중에서 메소드와 필드의 정보만을 이용하였다. Op 코드 정보는 표 5와 같다.

표 5에서와 같이 Op 코드로부터 다음과 같은 메트릭 정보를 얻어낼 수 있다. getfield와 setfield는 소스코드에서 필드에 대한 정보를 갖고 있어서 setfield로 선언된 필드 값은 constant_pool에 저장되어 언제든지 getfield를 통해 호출되어 재사용 되어 쓸 수 있다.

메소드에 대한 Op 코드는 메트릭 측정에서 다양한 정보를 얻어낼 수 있다. Java 소스코드에서는 호출된 메소드 중에서 상속관계나 사용자 선언 메소드, 정적(static) 메소드, 그리고 인터페이스 정보를 명확히 구별하기 어렵다. 이러한 문제점은 프로그램 이해 부분이나 메트릭 측정 부분에 있어서도 가장 어려운 문제로 남아있는데 자바 가상 머신에 의해 컴파일된 클래스 파일은 이러한 문제점을 Op 코드로 구별해주므로 문제 해결이 된다.

표 5 Java 클래스 파일의 Op 코드

Op 코드	기 능
Nop	"no operation(동작없음)"의 약자이다. 가상기계가 nop을 만날 경우 어떤 작업도 수행하지 않고 넘어간다.
스택에 값 밀어넣기	
const	상수값을 operand 스택에 저장한다. 형식 : typeconst_value 예 : iconst_2 (int형 정수 2를 넣는다.)
bipush sipush	스택에 부호화된 byte형 상수를 넣는다. bipush : 코드 배열에서 이 명령어 바로 뒤에오는 바이트값으로 동작한다. sipush : 스택에 부호화된 short형 상수를 넣는다.
ldc	"load constant(상수로드)"의 약자로 스택에 있는 상수를 상수풀로 복사한다.
필드에 대한 정보	
getfield	스택으로부터 레퍼런스를 꺼내오는 기능을 담당한다.
setfield	field의 값을 초기설정 해주는 기능을 담당한다.
메소드에 대한 정보	
invokevirtual	일반적 메소드 호출시 사용된다.
invokespecial	현재 객체의 상위 클래스에 존재하는 메소드를 호출할 때 사용한다.
invokestatic	정적 메소드 호출시 사용한다.
invokeinterface	인터페이스 영역에 정의되어 있는 메소드 호출시 사용한다

표 6 Op 코드로부터 추출할 수 있는 메소드 상속정보

```
<Source>
 ...
MenuItem aboutMenuItem = new MenuItem("About QAJava..");
       menu.add(aboutMenuItem);
       menubar.add(menu);
 ...

<Disassemble>
 ...
 418 new #69 <Class Java.awt.MenuItem>
 421 dup
 422 ldc #12 <String "About QAJava..">
 424 invokespecial #115 <Method Java.awt.MenuItem(Java.lang.String)>
 427 astore 22
 429 aload_1
 430 aload_22
 432 invokevirtual #125 <Method Java.awt.MenuItem add(Java.awt.MenuItem)>
 435 pop
 436 aload_2
 437 aload_1
 438 invokevirtual #124 <Method Java.awt.Menu add(Java.awt.Menu)>
  ...
```

본 시스템에서는 메소드에 관련된 대부분의 정보를 Op 코드에서 추출하였다. 그 예제는 표 6과 같다.

표 6에서 <Source>에 보여지는 코드는 Java 소스 프로그램의 한 단편으로서 MenuItem을 생성할 경우 그 클래스가 명확히 어디에서 상속받은 것인지 알 수 없다. 마찬가지로 add() 메소드도 명확히 어떤 클래스에 존재하는 메소드를 상속받았는지 알 수 없다. 따라서, 프로그램 이해함에 있어서 구체적인 상속이나 메소드 호출 관계를 이해하기 어렵고 정확한 정보를 추출할 수 없기 때문에 메트릭 측정에도 많은 어려움을 가져올 수 있다. 한편, <Disassemble> 부분에 보여지는 역어셈블된 코드는 각 클래스와 메소드에 대한 이미 컴파일된 .class 파일을 분석하여 동적인 정보를 추출하므로 명확한 상속 패키지명을 알 수 있고 메소드 호출도 명시적으로 정확한 클래스 위치를 지정해주므로 프로그램 이해에 도움을 줄 수 있으며 나아가 메트릭 측정시에도 메소드간의 관계나 클래스간의 관계를 명확히 규정할 수 있다. 따라서, 본 도구는 소스코드로부터 얻기 어려운 정보들을 이와 같이 역컴파일시킨 .class 파일에서 추출함으로써 보다 정확한 품질평가 도구에 적용시키고자 하였다. 본 시스템에서 중점적으로 추출한 정보는 메소드 정보로서 다음과 같다.

- invokespecial : invokespecial은 상위 클래스로부터 직접적으로 상속받은 클래스를 나타낼 때 사용하는 바이트코드 명령이다. 또한 import 패키지로부터 상속받아 new로 생성할 경우에도 이 명령이 사용된다.

- invokevirtual : invokesvirtual은 현재 클래스에서 선언된 메소드나 void, native형(int, flaot등) 메소드 사용시 사용되는 바이트코드 명령이다. 한편, [21]에 따르면 invokevirtual이 일반적 메소드를 호출하는 경우 사용되는 명령이라고 설명되어 있지만 정확하게 invokespecial과 구분하기는 어렵다. 하지만, invokespecial이 리턴형이 없는 점에 반해 invokevirtual은 리턴형이 존재하는 것으로 구별할 수 있다. 보통 선언된 리턴형의 종류를 보면 void, native형(int, float, char 형 등이 있다.)

- invokeinterface : invokeinterface는 직접적으로 클래스를 상속받는 Java의 extends와 인터페이스 상속을 하는 implements 명령을 구분해 주는 유일한 바이트코드 명령이다. Java 프로그램은 다중상속이 존재하지 않으므로 대부분 인터페이스를 통해 다중상속 기능을 사용하고 있다. 하지만 소스코드에서는 직접적 상속과 인터페이스 상속을 통해 사용한 메소드 구분이 명확치 않으므로 클래스 파일을 통해서만 그 명확한 정보를 얻어낼 수 있다.

- invokestatic : 정적 메소드는 프로그램 컴파일시 이미 메모리 할당에 필요한 정보를 호출해서 사용한 경우에 이용되는 명령으로 일반적인 메소드 호출과 구분된다.

위와 같은 네 가지의 바이트코드 명령은 역컴파일된 코드로부터 가장 정확하게 얻을 수 있는 정보로서 본 논문에서는 그림 1과 같이 이 정보를 프로그램 이해 지원모듈에 필요한 클래스 계층구조 및 메소드 호출관계

에 응용하였다. 먼저 Java 소스 프로그램을 javac로 컴파일하여 .class 파일로 변환한 후 원하는 정보를 추출할 수 있도록 작성한 파서(parser)를 통해 각 모듈별로 필요한 정보를 정보저장소에 저장한다. 저장된 정보는 각 모듈에서 원하는 결과를 호출할 때 사용자에게 보여준다

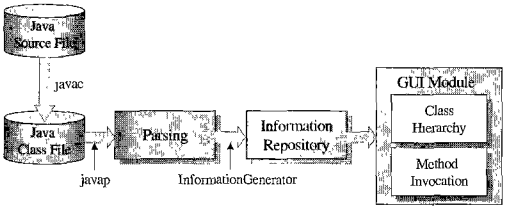

그림 1 바이트코드 정보추출 구성도

4. 시스템 설계 및 구현

본 장에서는 3장에서 제안된 방법에 의하여 구축된 품질평가를 지원하는 메트릭 측정 시스템을 설명한다. 제안한 시스템의 전체적인 기능과 특징은 다음 절에서 설명한다.

4.1 시스템 구조

본 시스템의 전체적인 구조는 그림 2과 같다.

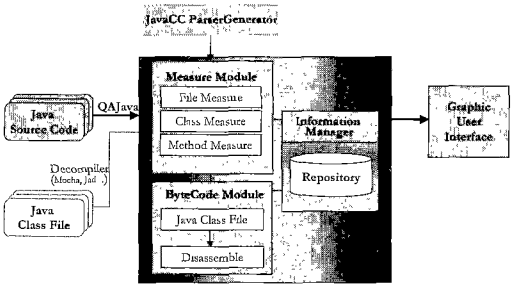

그림 2 시스템 구조

본 논문에서 사용한 도구 QA/Java(Quality Assurance Java)는 Java 파일을 분석하여 프로그램 이해와 품질평가에 필요한 메트릭 측정을 담당하는 시스템이다.

Java 소스 프로그램을 입력으로 받을 경우 먼저, 프로그램 구문분석을 수행한 후 필요한 추출을 위해 측정 모듈을 수행한다. 측정 모듈에서는 파일, 클래스, 메소드 단위로 정보를 추출하여 정보 저장소에 저장한다. 동시에 바이트코드로부터 필요한 정보는 역어셈블과정을 거친 다음 정보를 추출하여 정보 저장소에 저장된다. 반면에, Java 프로그램이 아닌 .class 파일을 입력으로 받는 경우 역컴파일러(decompiler)를 통해 Java 프로그램으로 변환된 다음 위와 마찬가지로 측정 모듈을 수행한다. 마지막으로, 정보 저장소에 저장된 추출 정보는 품질평가에 필요한 메트릭 값을 계산하는데 사용된다. 이 메트릭 값은 사용자 인터페이스를 통해 보다 쉽게 이해할 수 있도록 비주얼하게 제공됨으로써 소스 프로그램과의 비교 분석을 용이하게 한다.

4.2 시스템의 구성 모듈별 특징

QA/Java의 인터페이스 구성모듈은 크게 세 가지로 구성된다.

4.2.1 소스 모듈

소스 모듈은 측정하고자 하는 Java 소스프로그램과 역어셈블된 바이트코드 정보를 보여준다. 본 시스템은 품질평가 시스템 구축을 위한 필요한 하나의 서브 모듈로서 기존의 객체지향 메트릭을 소스프로그램에 적용시킨 메트릭 측정 시스템이다. 따라서 기존에 제안된 몇몇의 메트릭 측정 시스템과 같이 단순히 메트릭 값만을 보여주는데 그치지 않고[24, 27, 34] 소스 프로그램을 메트릭 값이 어떻게 얻어지는지 병행적으로 분석해야 하므로 소스코드를 보여준다. 또한 Java 언어의 특성상 소스 프로그램에서 얻을 수 있는 정보에 대한 제약이 따르기 때문에 역어셈블된 바이트코드를 사용하였다. 따라서 소스 프로그램뿐만 아니라 바이트코드의 정보도 보여줌으로써 분석을 용이하게 지원하고 있다.

4.2.2 프로그램 이해 지원모듈

프로그램 이해지원 모듈은 프로그램의 구조를 보다 빠르고 직관적으로 이해하도록 사용자에게 제공하는 기능이다. 일반적으로 객체지향 언어에 있어서 가장 분석하기 어려운 부분이 상속개념이다. 특히 Java 언어와 같이 클래스를 import하여 프로그램 구현을 하게되면 상속받은 클래스에 대한 정보를 얻기 힘들게 되므로 사용자가 쉽게 이해할 수 있는 이해정보가 필요하다. 본 시스템이 지원하는 이해모듈은 다음과 같다.

① 하나의 파일에 대한 클래스와 메소드 정보를 보여주는 클래스 브라우저
② 클래스 내에서 사용되고 있는 메소드들간의 호출관계
③ 현재 클래스에 대한 상속관계
④ 클래스 계층구조와 생성자, 메소드 정보등 상세한 문서화(documentation)를 지원하는 JavaDOC

4.2.3 품질평가 모듈

품질평가 모듈은 두 가지로 분류된다. 먼저 소스프로그램 측정 모듈로 기존에 제안된 메트릭 연산이 아닌

소스코드로부터 직접적으로 얻어지는 메트릭 값을 측정하여 보여준다. 그 정보들은 기존에 객체지향 메트릭로 제안된 OOMetrics[10]값과 Java 언어의 특성상 필요한 추가적인 값들로 구성되었다. 소스코드로부터 직접적으로 측정되는 값은 파일, 클래스, 메소드 단위로 이루어지며 그 내용의 일부는 표 7과 같다. 두 번째는 통합적 품질평가를 지원하기 위해 Briand가 정의한 속성을 만족하는 메트릭 측정값을 제공한다. 이 값은 크기, 길이, 복잡도, 결합도, 응집도로 각각 구성되며 각 메트릭값은 테이블과 파이 그래프로 보여진다.

표 7 소스코드로부터 얻은 측정값

측정단위	측 정
파일	파일 이름 클래스의 개수 import 개수 패키지 이름
클래스	인터페이스인지 클래스인지 구분 클래스/인터페이스 이름 생성자 수 필드 수 메소드 수 패키지 이름
메소드	클래스 이름 최대 중첩 정도 멤버 타입 연산자 수 블록의 개수 분기문의 개수 형변환(cast)의 수 식(expressons)의 수 매개 변수(formal parameter)의 수 함수 호출(function call)의 수 리터럴(literal)의 수 토큰의 개수 ...

4.2.4 도움말 모듈

도움말 모듈은 본 시스템에서 선택한 파일에 대한 이해를 쉽게 할 수 있도록 지원하는 모듈로서 JavaDOC과 각 메트릭에 대한 구체적인 설명을 제공한다.

4.3 클래스 다이어그램

도구 구축을 위해 작성된 클래스 다이어그램은 그림 3과 같다.

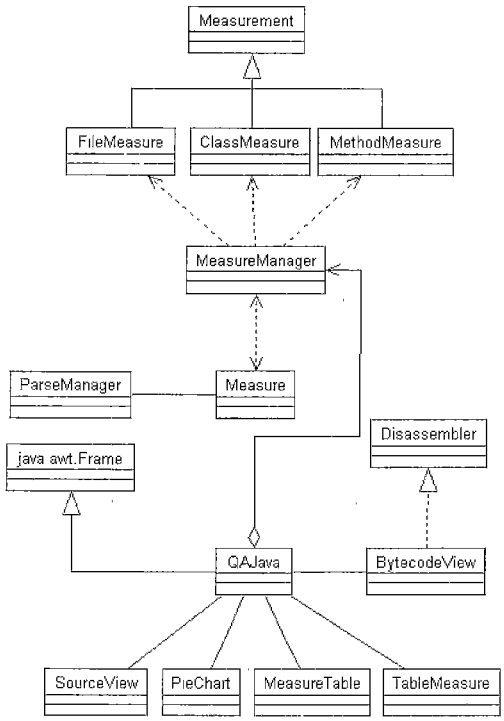

그림 3 도구의 클래스 다이어그램

다이어그램은 구문분석을 통해 구축된 메트릭 측정 부분과 바이트코드를 파싱해서 보여주는 부분, 마지막으로 각 측정값과 분석값을 사용자 인터페이스를 통해 보여주는 부분으로 구성된다.

또한 도구를 통해 분석된 데이타의 자료는 서로 패키지, 파일, 클래스 단위로 이루어진 링크드리스트(linked list)로 연결되어 있고 그 파일에 대한 측정값은 이진트리로 저장하여 삽입과 검색이 편리하도록 구성되어져 있다. 마찬가지로 하나의 파일에는 여러 개의 클래스가 존재할 수 있어 그 클래스들은 다시 링크드리스트로 연결되어 있고 자체의 측정값은 이진트리로 구성되어 있다. 이와 같은 방법으로 클래스내의 메소드와 멤버변수도 링크드리스트 연결과 자체의 이진트리 구조들을 가지고 있는 반복 구조(iteration)로 이루어진다[22].

이러한 자료구조는 프로그램 분석과정에서 분석되는 값들의 삽입, 정렬이 편리하게 이루어지고 메트릭 측정과 프로그램 이해모듈을 위한 정보 추출시에도 원하는 정보를 빨리 찾을 수 있다.

본 도구의 실행 초기화면은 다음과 같다.

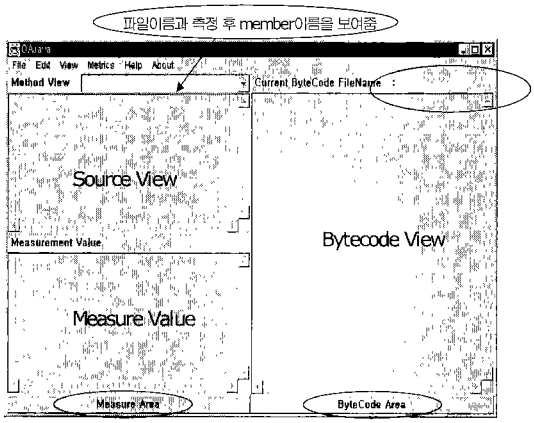

그림 4 실행 초기 화면

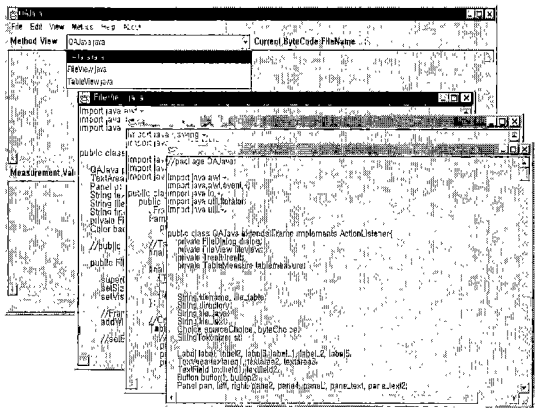

그림 5 File 메뉴에서 Open 선택 화면

그림 4에서 Source View 영역은 Java 소스코드를 보여주고, ByteCode View 영역에서는 아래의 "ByteCode Area" 버튼을 클릭할 경우 바이트코드로 구성된 클래스 파일을 역어셈블하여 정보를 보여준다. 마지막으로 Measure Value 영역은 "MeasureArea" 버튼을 클릭할 경우 Java 소스코드 분석후 얻은 측정값을 프로젝트, 클래스, 멤버 단위로 보여준다.

5. 실행 결과 및 도구의 평가

5.1 실행 결과

3장과 4장에서 제안된 방법을 이용한 시스템 개발 환경은 다음과 같다. Java 소스 프로그램의 구분 분석은 JavaCC 0.8pre2를 사용하였고 언어는 JDK 1.3, 그리고 프로그램의 역컴파일러로는 JAD[23]를 사용하였다.

그림 5에서 그림 9는 개발된 시스템을 실행하였을 때 사용자에게 보여지는 정보 중 몇 가지 예를 보여준다.

그림 5는 File 메뉴의 Open 버튼을 클릭한 후 파일을 불러온 결과이다. 오픈된 파일은 초이스(choice) 박스로 파일의 이름이 누적되면서 소스 창이 여러 개 나타난다.

그림 6은 초이스 박스에서 측정하고자 하는 파일을 선택하여 분석한 결과이다. 도구 맨 아래의 "Measure Area"의 버튼을 클릭하게 되면 프로그램 분석단계를 거친 후 소스코드로부터 측정값을 얻어내어 "Measure View"로 보여준다.

마찬가지로 "ByteCode Area" 버튼을 클릭하면 선택된 파일에 대한 바이트코드를 역어셈블한 결과를 보여준다. 마지막으로, "ClassBrowser"는 파일에 대한 클래스, 메소드, 멤버 구조를 트리 형태로 보여준다.

그림 6 메트릭 분석 후 View 메뉴에서 ClassBrowser 선택 화면

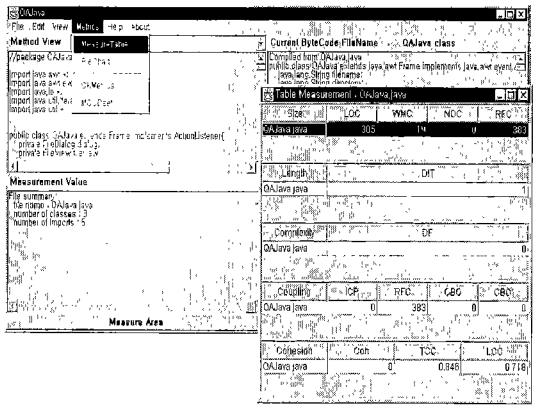

그림 7 Metrics 메뉴에서 MetricsTable 선택 화면

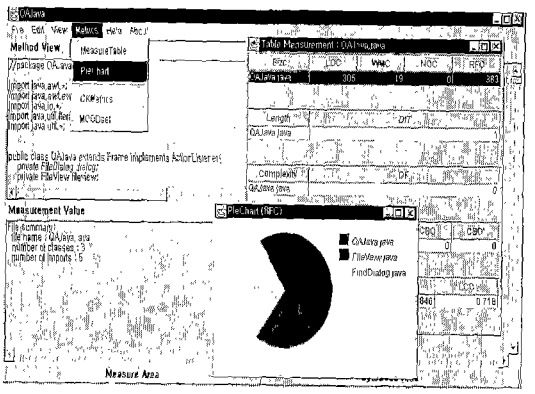

그림 8 Metrics 메뉴에서 PieChart 선택 화면

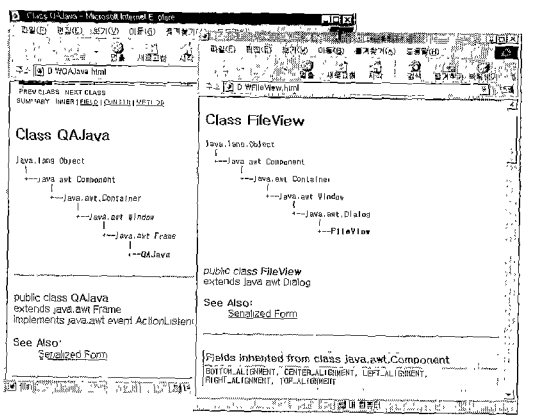

그림 9 Help 메뉴에서 JavaDOC의 선택 화면

그림 7은 메트릭 모듈로서 "Metrics" 메뉴에 있는 "MeasureTable"을 실행한 화면이다. 측정 단위는 크기, 길이, 복잡도, 결합도, 응집도 단위로 이루어지고 각 메트릭에 대한 결과를 보여준다.

그림 8은 파이차트로서 메트릭 테이블에서 보여준 메트릭 값을 여러 개의 파일과 비교 분석하기 위해 제공되는 차트이다. 현재 보여준 예제에서는 3가지 파일에 대한 RFC의 값을 보여준다.

그림 9는 Help 메뉴에서 제공되는 JavaDOC을 보여준다. JavaDOC은 Java 프로그램의 각 파일에 대한 문서화를 제공하여 프로그램의 전반적인 구조와 생성자, 메소드등의 상세사항을 이해할 수 있게 제공한다.

5.2 도구의 평가

객체지향 소프트웨어 품질평가에서 필요한 자동화 도구는 복잡한 소스코드의 이해와 직관적 비교·분석을 위해서는 반드시 필요하다. 각각의 도구는 프로그래밍 언어, 관점, 측정 기준에 되는 메트릭에 따라 서로 다른 측정값을 보여주므로 이 장에서는 기존의 객체지향 소프트웨어 품질평가 도구와 본 논문에서 제안한 도구의 특성을 비교·평가하고자 한다.

PR:QA사에서 개발한 Software Quality Assurance C++(이하 QA/C++)는 C++를 대상으로 한 분석 시스템으로서 프로그램 이해와 메트릭 측정을 동시에 지원하는 품질평가 시스템이다[24]. QA/C++는 크게 품질평가 모듈 프로그램 이해 모듈로 이루어져있다. 품질평가 모듈은 C++ 소스 프로그램으로부터 파일(file), 클래스(class), 함수(function) 단위의 측정값을 얻어내어 ISO 표준에서 정한 기준에 따라 품질평가 결과를 보여주고 분석된 평가 결과의 비교 분석을 위해 히스토그램이나 Kiviat 다이어그램과 같은 그래픽 모듈을 제공한다. 프로그램 이해 모듈은 class/extern의 호출관계와 Fan-in/Fan-out 분석을 보여주는 관련성 브라우저(relationships browser)와 분석 코드에서 함수의 구조를 시각적으로 보여주는 함수 구조(function structure)를 제공하고 있다.

BANDA[25]는 Java 소스 프로그램에 대한 메트릭 측정값을 보여준다. 이 시스템은 Java 소스 프로그램 구분분석(Parsing)에 필요한 JavaCC(Java Compiler Compiler)[26]를 이용하여 소스 프로그램 분석한 다음 그 과정에서 생성된 파일에 사용자가 원하는 코드를 삽입함으로써 측정값을 얻어내도록 구축되었다. 이 시스템은 소스코드로부터 품질평가에 필요한 '직접적인 측정'[1] 값을 얻어낸다. 측정은 프로젝트(project), 파일(file), 멤버(member) 단위로 이루어진다. 또한, 각 메소드(method)에 대한 측정값과 관련 소스코드가 동시에 보여지기 때문에 프로그램 분석이 간편하다. 하지만, 소스코드로부터 얻을 수 있는 측정값을 상세히 알 수 있지만 단지 직접적 측정값만을 나열하기 때문에 프로그램 평가를 하기 위해서는 이 값을 다시 통계·비교 분석하여 결과를 얻어내야 하는 불편함이 있다. 그러므로, 사용자가 통합된 품질평가를 얻을 수 있는 간접적 측정이 필요하고, 나아가 그 결과에 대한 직관적 이해를 할 수 있도록 테이블이나 그래프와 같은 시각화 모듈이 필요

1) "직접적 측정" 방법은 소스코드로부터 직접 얻어내는 정보를 말한다. 이것은 코드로부터 다양하게 발생한 빈도 수에 의해 측정되며, 그 예로는 라인 수, 클래스의 수, 메소드의 수가 있다. 반면에 "간접적 측정" 방법은 "직접적 측정"으로부터 얻어진 정보를 바탕으로 평균, 변량, 편차, 분포등과 같은 표준화된 통계적 변수를 추출하는 방법이다. 이것은 품질평가에 있어서 집합적 정보를 보여줌으로써 비교·분석을 도와준다[27].

하다.

Andrew Cain과 Rajesh Vasa가 개발한 JMetric[28]은 BANDA와 마찬가지로 Java 소스코드에 대한 품질평가 시스템으로서 직접적 측정값을 수치적으로 보여주는데 그치지 않고 측정된 값을 사용자가 직관적으로 이해하도록 그래픽 모듈을 지원하고 있다. 또한, 소스코드로부터 얻어진 측정값을 통합하여 보다 통합적 품질평가를 얻을 수 있도록 제공하였다. JMetric은 소스코드로부터 프로젝트, 패키지, 클래스, 메소드, 변수 단위로 직접적 측정값을 얻어내는 측정 모듈과 소스코드로부터 얻은 측정값을 사용자가 직관적으로 이해하며 비교, 분석할 수 있도록 프로젝트와 클래스 단위의 챠트를 보여주는 챠트 기능, 측정을 통해 얻어진 정보와 품질평가에 내용을 문서화할 수 있도록 챠트와 측정값을 출력해주는 보고서 기능, 그리고 측정값을 테이블화하여 보여주는 테이블 기능과 프로젝트의 구조를 트리 구조로 보여주는 트리 기능을 제공하고 있다.

위에서 설명한 JMetric은 기존에 개발된 메트릭 도구에서 보여준 단순한 정량적 측정 뿐만 아니라 사용자에게 직관적 이해를 돕기 위한 시각적 도구를 제공함으로써 보다 사용하기 편리하다. 하지만, 프로그램 이해를 위한 분석 모듈이 지원되지 않아 소스 프로그램의 구조를 이해하지 못한 채 메트릭 측정을 하게 되므로 분석 도구로서는 불편한 점이 있다. 또한, 일정한 평가 기준이 없이 선택된 메트릭을 사용하기 때문에 일반화가 어렵다.

이 외에도 Java 프로그램의 이해와 메트릭을 생성하는 도구[32]가 몇몇 연구자에 의해서 개발된 경우는 있지만 본 논문에서는 수학적으로 정형화된 품질평가 기준과 이 기준을 기반으로 통합된 객체지향 소프트웨어 메트릭 적용함으로써 Java 프로그램에 적합한 메트릭을 검증하고 수정 및 제안하는데 보다 명확한 이론적 기반을 제공하고자 한다.

표 8은 지금까지 살펴본 품질평가 시스템들과 본 논문에서 구현한 도구들의 특성을 비교한 결과이다.

위의 시스템 중에서 Java 소스 프로그램을 대상으로 직관적 이해를 동반한 품질평가 메트릭 측정과 프로그램 이해를 동시에 제공하는 시스템은 존재하지 않는다. JMetric이 시각적 평가지원 모듈 포함하여 메트릭 측정을 지원하지만 소스 프로그램을 이해할 수 있는 이해모듈이 제공되지 않기 때문에 메트릭 검증을 필요로 하는 분석자 입장에서는 어려움이 있다. 또한, 각 시스템들이 사용한 측정 기준을 살펴보면 같은 객체지향 언어임에도 불구하고 서로 다른 기준을 가지고 측정하였으며 평가 방법도 다양하다. 따라서 본 논문에서 구축한 시스템은 Briand가 제안한 속성기반을 토대로 분류된 메트릭을 Java 소스 프로그램에 적용시켰다. 나아가 평가 결과에 대한 직관적 분석을 도울 수 있는 이해 모듈로써 파이 차트와 테이블 제공하였으며 동시에 프로그램 이해를 도울 수 있도록 클래스 계층구조와 소스 뷰, 그리고 메소드 호출관계를 모듈을 제공하였다.

표 8 도구의 특성 비교

특성 / 시스템	적용 언어	측정 기준	평가방법	시각적 평가지원 모듈	프로그램 이해지원 모듈
QA/C++	C++	파일 클래스 함수	ISO 표준지향	히스토그램 Kiviat 다이어그램	관련성 브라우저 함수 구조
BANDA	Java	프로젝트 파일 멤버	정량적 평가 직접적 측정 값	정량적 수치만 보여줌	소스 뷰
JMetric	Java	프로젝트 패키지 클래스 메소드 변수	정량적 평가 간접적 측정 값 직접적 측정 값	챠트 테이블 트리구조	없음
QA/Java	Java	패키지 클래스 메소드	정량적 평가 직접적 측정	파이 차트 테이블	소스 뷰 클래스 계층구조 메소드 호출관계

6. 결론 및 향후 연구

본 논문에서는 프로그램의 재사용 및 유지보수 관점에서 반드시 필요한 Java 프로그램의 품질평가를 위해 기존의 객체지향 메트릭의 타당성 및 검증에 필요한 자동화된 메트릭 측정 시스템을 구축하였다. 이 시스템의 기능을 요약하면 다음과 같다.

· Java 소스 프로그램으로부터 품질 평가에 필요한 직접적 측정값을 프로젝트, 클래스, 멤버 단위로 분석하였다.

· 기존의 객체지향 메트릭을 수학적 검증하여 속성별로 분류한 Briand의 속성 기반 메트릭을 기반으로 Java 프로그램을 평가하였다.

· Java 소스 프로그램에서 얻기 어려운 상속, 클래스, 메소드 호출관계, import 클래스등의 정보를 바이트코드 역어셈블 과정에서 추출하여 품질평가와 이해 모듈에 적용하였다.

· Java 소스 프로그램의 구조적 이해를 위한 프로그램 이해 모듈을 제공하였다.

· 품질평가를 지원하는 측정시스템으로서 테이블이나 파이차트와 같은 프로그램 분석 모듈을 제공하여 메트릭에 대한 비교, 분석을 직관적으로 이해할 수 있도록 제공하였다.

본 시스템은 Java 프로그램에 대한 품질평가 시스템 개발을 위한 지원 모듈로서 Java 프로그램의 소스 분석과 기존 메트릭에 대한 타당성 분석을 도와준다. 또한, 분석과정에서 필요한 프로그램의 직관적 이해를 돕는 기능을 제공하여 품질평가 시스템 구축을 위해 우선적으로 필요한 기능을 포함하고 있는 시스템이다. 따라서, 이미 산출된 Java 프로그램을 기존 메트릭에 적용시킴으로써 타당성과 문제점을 유도해 낼 수 있고 메트릭에 대한 분석이 완료되면 각 평가 기준에 적합한 품질평가를 마련할 수 있으며, 나아가 Java 언어의 특성에 필요한 보다 구체적이고 명확한 품질 메트릭을 제안의 기반을 제공할 수 있게 될 것이다.

향후 연구과제로는 완전한 품질평가 시스템을 위해 필요한 메트릭 모듈을 보완하고 프로그램 분석 위해 다이어그램 모듈의 추가가 필요하다. 또한 아직 제공되고 있지 않는 통계 모듈을 추가시킴으로써 전체 통합적 분석이 가능하도록 하여야 하며 나아가 프로그램 이해는 품질평가 시스템 구축에 있어서 중요한 기능 중 하나이므로 사용자에게 직관적 이해를 도울 수 있도록 이해 모듈의 확장이 필요하다.

마지막으로 최근 프로그램의 컴포넌트화가 쟁점화되고 EJB(Enterprise JavaBeans)와 같은 서버측 애플리케이션(server-side application)의 구축이 활성화되면서 프로그램 이해와 재사용은 프로그램 품질측정에 있어서 보다 중요한 역할을 하고 있다[1]. 따라서, 기존의 객체지향 소프트웨어 메트릭을 보다 확장하여 컴포넌트 및 서버측 애플리케이션에서도 충분히 적용 가능하도록 분석 및 검증을 할 필요성이 있다.

참 고 문 헌

[1] David M. Arnow, and Gerald Weiss, *Introduction to Programming Using Java*, Addison Wesley, 1998.

[2] Ed ROMAN, *Mastering EJB and the Java 2 Platform, Enterprise Edition*, Wiley, 1999.

[3] Lionel C. Briand L., and Morasca S., "Property-Based Software Engineering Measurement," *IEEE Trans. on Software Eng.*, Vol. 22, No. 2, January. 1996.

[4] Briand L., Daly J., and Wust J., "A Unified Framework for Cohesion Measurement in Object-Oriented Systems," *Empirical Software Engineering Journal 3(1)*, pp. 56-117, 1998. Also avaliable as Technical Report ISERN-97-05.

[5] Briand L, Daly J., Porter V., and Wust J., "A Comprehensive Empirical Validation of Product Measures for Object-Oriented Systems," *Technical Report ISERN-98-07*, 1998.

[6] Roger S. Pressman, *Software Engineering-A Practitioner's Approach*, McGraw-Hill, 4th., 1998.

[7] Bieman J. M., and Kang B. K, "Cohesion and Reuse in an Object-Oriented System," in Proc. *ACM Symp. Software Reusability(SSR '94)*, pp. 259-262, 1995.

[9] Bush M. E. and Fenton N., "Software Measurement: A Conceptual Framework," *The Journal of Systems and Software*, Vol.12, No.3, pp. 223-231, 1990.

[10] Lorenz, *Object-Oriented Software Metrics: A Practical Guide*, Prentice Hall, 1994.

[11] Fenton N., *Software Metrics, A Rigorous Approach*, Chapman and Hall, London, 1991.

[12] Chidamber S. R, and Kemerer C. F., "Towards Metrics Suite for Object Oriented Design," in A. Paepcke,(ed.) *Proc. Conference on Object-Oriented Programming: Systems, Languages and Applications (OOPSLA'91)*, October 1991.

[13] Chidamber S. R, and Kemerer C. F., "A Metrics Suite for Object Oriented Design," *IEEE Trans. on Software Eng.*, 20(6), pp.476-493, 1994.

[14] Harrison R., and Steve J. Counsell, "An Evaluation pf the MOOD set of Object-Oriented

Software Metrics," *IEEE Trans. on Software Eng.*, Vol.24, No.6, pp. 491-496, June 1998.
[16] Weyuker E. J., "Evaluating Software Complexity Measures," *IEEE Trans. Software Eng.*, Vol.14, No.9, pp. 1357-1365, September. 1998.
[17] Oveido E. I., "Control Flow, Data Flow and Program Compleixty," Proc. *IEEE COMPSAC*, pp. 146-152, November. 1980.
[18] Lee Y. S., Liang B. S., Wu S.-F., and Wang F. J., "Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow," in Proc. *International Conference on Software Quality*, Maribor, Slovenia, 1995.
[19] Briand L., Morasca S., and Basili V., "Defining and Validating High-level Design Metrics," CS-TR 3301, *Univ. of Maryland, College Park, Md. Submitted for publications.*
[20] Elliott Rusty Harold, *Java Secrets*, UDG Books, 1997.
[21] Bill Venners, *Inside the Java Virtual Machine*, McGraw-Hill, New York, 1998.
[22] Mitchell Waite, and Robert Lafore, *Data Structure and Algorithms in Java*, Gilbert & McCarty, 1998.
[23] Pavel Kouznetsov, *JAD v1.5.5.3*, http://web.unicom.com.cy/~kpd/jad.html
[24] Brian W. Bush, *BANDA Java Packages*, http://www.sladen.com/Java
[25] Sun Microsystems, *Java Compiler Compiler*, http://www.suntest.com/JavaCC
[26] Jagdish Bnasiya, and Carl Davis, *Automated Metrics and Object-Oriented Development*, Dr. Dobb's Jorunal, Dec. 1997.
[27] Andrew Cain, Rajesh Vasa, *JMetric*, http://www.csse.swin.edu.au/cotar/jmetric/index.html
[28] 이선아, 최병주, "소프트웨어 개발 과정에서 제품의 품질 척도를 적용하는 방법", 정보과학회논문지 : 소프트웨어 및 응용, 제27권 3호, pp. 217-226, 2000. 3.
[29] Berard, Edward V., "Metrics for Object-Oriented Software Engineering," an Internet Posting on Comp. Software-Engneering, Jan. 28, 1995.
[30] Binder, Robert V., "Testing Object-Oriented Systems:A Status Report," American Programmer, Vol.7, No.4, pp. 22-29, 1994.
[31] Ejiou, L., Sofrware Engineering with Formal Metrics, QED Publishing, 1991.
[32] 정지환, 황선명, "자바 프로그램의 그래픽 구조 분석과 메트릭스 생성도구의 설계", 정보과학회 가을 학술발표논문집, 제26권 2호, 1999.
[34] QA/C++, QA/C++ Version 3.1 Documentation, PR:QA, 1996.

박 옥 자
1999년 8월 전북대학교 전산통계학과 석사. 1999년 9월 ~ 현재 전북대학교 전산통계학과 박사과정. 관심분야는 소프트웨어공학, 에이전트 공학, 컴포넌트기술, 분산객체기술 등임

유 철 중
정보과학회논문지 : 컴퓨팅의실제 제 7 권 제 1 호 참조

장 옥 배
정보과학회논문지 : 컴퓨팅의실제 제 7 권 제 1 호 참조