□기술해설□

객체지향 시스템의 복잡도 측정

기전여자전문대학 유철중*
전북대학교 김용성* · 장옥배*

목 차

1. 서 론

소프트웨어 개발을 공학적 과정으로 생각한다면 소프트웨어 프로세스(process)와 산출물(product)을 우리가 보다 잘 이해할 수 있고 평가, 예측, 그리고 제어할 수 있는 측정(measurement) 요소가 필요하다. 이러한 프로세스 또는 산출물의 성질을 설명하려는 척도(metrics) 또는 모델을 이용한 측정은 소프트웨어 개발과정의 중요한 부분으로서 인식되어 왔다. 소프트웨어 측정 영역은 지속적으로 발전되어 왔으며 측정의 범위, 측정이 적용될 수 있는 관점의 이해, 측정의 정의와 해석에 대한 체계, 척도들이 근거를 두고 있는 측도(measures)와 모델의 제안, 측정의 자동화, 그리고 조직(기업체)에서의 측정의 응용등과 같은 분야에서 주로 연구가 이루어졌다.

소프트웨어 위기를 해결하기 위한 방법론으로 대두된 객체지향 기법(object-oriented technology)은 소프트웨어공학 뿐만 아니라 데이타베이스, 인공지능, 운영체제, 사용자 인터페이스, 시뮬레이션 등 컴퓨터 과학의 여러 영역에 도입되어 활발히 연구되고 있다. 이러한 다양한 영역에서 객체지향 기법은 실질적인 개발 생산성 증가를 가져왔다. 그러나 공식적으로 이러한 생산성 증가를 규명하고 인지 메카니즘(cognitive mechanisms)을 식별하는 일은 연구 과제로 남아있다[Taen 89]. 이러한 새로운 패러다임(paradigm)하에서 소프트웨어 개발을 모델링하는 연구는 그렇게 많지 않다[Jens 91, Lara 90]. 따라서 객체지향 시스템에 적합한 모델 및 함축적인 복잡도 척도의 개발이 필요하다[Moreu87, More 90a, More 90b]. 이러한 종류의 모델 및 척도의 개발과 확인은 체계적으로 수집된 다량의 구조 및 생산성 데이타가 필요하나 체계적으로 수집된 개발 환경에 대한 정보가 현저하게 부족한 상태이며 이에 대한 연구도 많지 않다. 또한 성능 데이타와 소프트웨어 특성의 수집이 신뢰할 수 있어야 하고 객체지향 시스템을 위한 소프트웨어 복잡도 척도의 연구가 장기간 필요하다. 이러한 복잡도 연구를 통해서 객체지향 환경이 다양한 응용 분야에서 생산성을 증가시키는데 어떠한 이유로 그처럼 효율적

*종신회원

인지를 규명할 수 있게 될 것이다.

객체지향성에 관점을 둔 척도는 객체지향 접근에 생소한 개발 요원들에 대한 학습 도구(learning tool)로서 사용될 수 있을 뿐만 아니라 구현의 객체지향 정도를 평가하는데 도움을 줄 수 있다[Bowm 90]. 또한 궁극적으로 조직의 소프트웨어 설계 표준을 정하는데 있어서 유용한 객관적 기준이 될 수도 있다[Chid 91, Chid94].

본 고의 2장에서는 소프트웨어 복잡도의 개념에 대해서 간단히 알아보고, 3장에서는 복잡도 관점에서 객체지향 소프트웨어의 특성을 논하고 척도 적용 실험결과를 보인다. 4장에서는 객체지향 패러다임의 복잡도와 관련된 연구동향과 세부 연구내용에 대해서 살펴보고, 5장에서 결론을 맺는다.

2. 소프트웨어 복잡도

소프트웨어의 품질 특성(characteristics)을 정량적이고 객관적으로 측정하기 위한 노력중의 하나가 소프트웨어 복잡도 연구이다. 소프트웨어 복잡도는 계산적 복잡도와 심리적 복잡도라는 두 종류의 복잡도로 나누어 볼 수가 있는데, 계산적 복잡도는 알고리즘의 복잡도를 계산시간(computing time)과 기억 장소 용량(memory space)으로 표시한 것이고, 심리적 복잡도는 프로그램을 작성하는 데 드는 노력정도 및 작성된 프로그램을 이해하는데 드는 노력정도와 밀접한 관련을 갖는다[Wagu 87, Zuse 90]. 일반적으로 복잡도 연구는 심리적 복잡도에 대한 연구로서 심리적 복잡도는 관리자나 프로그래머의 직관적 관점(intuitive view)에 의한 프로그램 복잡도이다.

소프트웨어 복잡도는 소프트웨어 개발 주기 전반부를 통해 테스트 노력 및 프로그래밍 시간의 측정, 프로그래머의 디버깅 능력 및 생산성 평가, 프로그램의 비구조도 측정, 신뢰성과 유지 보수성 및 소프트웨어의 요구 사항에 대한 예측 등 매우 값진 정보들을 제공해 주며, 프로그램으로부터 직접 측정되는 복잡도는 앞으로의 테스트에서 발견될 에러 수의 예측 및 프로그램의 변경에 소요될 노력을 예측하는데 사용된다.

3. 객체지향 패러다임과 복잡도 측정

3.1 객체지향 소프트웨어공학과 척도

개발을 통제하는 필수적 방법은 척도를 사용하는 것인데 그러한 척도는 개발 과정이나 산출물의 여러 측면을 측정할 수 있다. 소프트웨어공학에서의 척도는 오랫동안 연구, 논의되어 왔지만 산출물이나 프로세스의 질을 향상시키는 방법으로서 널리 사용되지 못했다. 소프트웨어 척도에서의 실제적 문제점은 측정하려고 하는 것을 정확히 측정할 수 없다는 점이며 따라서 우리가 측정할 수 있는 것과 측정하려고 하는 것 사이에 관계가 있음을 가정하여야 한다. 프로세스 관련 척도(process-related metrics)가 산출물 관련 척도(product-related metrics)보다 훨씬 더 많이 사용되어 왔기 때문에 먼저 프로세스 관련 척도에 대하여 살펴본 후 산출물 관련 척도에 대하여 알아본다.

3.1.1 프로세스 관련 척도

프로세스 관련 척도는 월당 소요인원, 스케줄 시간, 테스트하는 동안의 고장 횟수 등과 같은 것들을 측정한다. 객체지향 소프트웨어공학(Object-Oriented Software Engineering; OOSE)과 같은 개발 과정을 조작하고 관리하기 위해서는 가능한 한 체계적으로 이러한 척도들에 관련된 데이타를 수집하는 것이 중요하다. 다음은 여러 프로세스 관련 척도들인데 OOSE에 의하여 작업할 경우 수집해야 하는 것들이다[Jaco 92].

1) 총 개발 시간

2) 각 프로세스와 서브프로세스의 개발 시간

3) 이전 프로세스로부터의 모델을 수정하는데 걸린 시간

4) 모든 종류의 서브프로세스에서의 소비 시간

(서브프로세스의 예)

사용 케이스(use case) 명세, 객체 명세, 사용 케이스 설계, 블록 설계,

블록 테스트, 각각의 특정 객체에 대한 사용 케이스 테스트

5) 검토 과정에서 발견된 서로 다른 고장 횟수

6) 이전의 모델에 관한 제안을 변경하는 횟수
7) 품질 보증 비용
8) 새로운 개발 프로세스와 도구의 도입 비용

이러한 척도들은 개발 프로젝트의 향후 계획에 대한 기초를 제공할 수 있다.

예를 들어, 하나의 사용 케이스를 명세서에 기입하기 위한 평균 시간과 명세되어야 할 갯수를 안다면 모든 사용 케이스를 명세서에 기입하기 위한 시간을 예측할 수 있다. 평균과 같은 통계적 척도들은 항상 표준 편차와 같은 척도의 신뢰성이 수반되어야 한다. 그렇지 않으면 예측의 정확성을 믿을 수 없을 것이다. 이러한 척도들은 프로젝트나 조직 또는 응용(application)이 상이하거나 스텝이 다를 경우 크게 달라진다는 점을 유의하여야 한다. 그러므로 환경을 고려하지 않고 현재의 데이타에 관한 일반적 결론을 내는 것은 위험천만한 일이다.

3.1.2 산출물 관련 척도

산출물 관련 척도로는 여러 종류의 척도가 제안되었다. 이들 중 어느 것도 일반적인 품질 예측자로서 유용한 것으로 받아들여지지는 못했다. 그러나 몇몇 품질 평가 기준은 하나의 어떤 품질과 관련된 성질을 예측하는데 사용될 수 있다.

코드를 포함한 산출물에 관한 전통적인 척도는 한 객체지향 소프트웨어에 있어서도 어느 정도 사용될 수 있다. 그러나 가장 일반적인 척도인 SLOC는 실제로 객체지향 소프트웨어를 측정하기에는 상당한 거리가 있다. 작성한 코드가 적을수록 코드가 더 재사용되었음을 의미하며, 이는 항상 그러한 것은 아니지만 산출물의 품질을 훨씬 높이는 결과를 가져다 준다. 상속성과 다형성은 객체지향 시스템에서 복잡도를 측정하는데 있어서 SLOC의 유용성에 관한 몇 가지 의문을 제기한다. 예를 들어, 상속되는 하나의 인스턴스 메소드의 SLOC를 그것을 상속하는 계층구조내의 모든 객체들의 부분으로서 카운트할 것인지 또는 그 메소드를 사용하는 객체들의 부분으로서 카운트할 것인지, 그렇지 않으면 정의 수준에서 단지 한번만 카운트할 것인지가 문제이다. 이와 더불어 클래스의 SLOC와 인스턴스 메소드의 SLOC를 다르게 구별하여 카운트할 것인지도 문제이다. 다형성이 사용될 때 의문은 해당 메소드의 SLOC가 그 메소드의 각 정의에서의 SLOC의 합계인지, 또는 각 정의는 다른 정의와 독립적으로 측정되어야 하는 새로운 메소드로서 보아야 하는지 등이다[Tega 92].

한편, 소프트웨어 과학 척도와 관련된 객체지향 시스템의 주요 의문은 카운트할 수 있는 연산자와 피연산자가 각각 무엇인지이다. 일반적으로 객체지향 시스템에서 메소드는 연산자로, 변수는 피연산자로 가정한다. 또한 메시지의 객체 부분은 단지 참조될 메소드를 정의하는 역할을 한다고 가정한다. 그러므로 하나의 객체는 연산자도 피연산자도 아니다. SLOC를 카운트하는 것과 관련된 문제들과 유사한 문제들이 소프트웨어 과학 척도 적용시에도 발생한다. 상속된 클래스/인스턴스 메소드를 그것을 상속하는 각 객체내에서 카운트할 것인지, 그것을 사용하는 객체들을 카운트할 것인지, 아니면 단지 그것을 정의한 객체내에서만 카운트할 것인지의 문제이다. 또한 하나의 상속된 변수는 어떠한 것인지, 한 메소드에 대한 각 다형적 정의는 하나의 독립적 연산자인지 또는 객체 계층구조내의 최상위 레벨에서 그 연산자의 카운트에 대한 증가를 의미하는지 등의 문제가 그것이다.

다른 일반적인 원시코드 척도들은 유용성 정도에는 차이가 있지만 객체지향 소프트웨어에 적용될 수 있다. 예를 들어, McCabe의 사이클로메틱 복잡도[McCa 76]는 한 그래프의 복잡도를 측정한다. 이 방법은 하나의 프로그램이 모든 가능한 경로를 가진 하나의 그래프로서 구성될 수 있도록 순서를 찾는 방법이다. '연결선의 수－노드수 ＋ 2' 로 계산되어지는 사이클로메틱 수는 작성한 프로그램이 어느 정도 복잡한 지를 나타내는 수가 된다. 프로그램의 복잡성으로 인해 에러의 유발 가능성이 증가하기 때문에 McCabe 수가 너무 크게 되면 좋지 않다. 일반화된 연구 결과에 의하면 어떠한 모듈도 McCabe 수가 10이상 되지 않도록 요구하고 있다. McCabe 수는 또한 경로 테스트를 하기 위한 테스트 케이스(test case) 수를 제공한다는 점을 주목하여야 한다.

위에서 정의된 것처럼 McCabe 수는 객체지

향 소프트웨어에서는 관계가 적을 것이다. 여기에는 여러 가지 이유가 있는데 그 첫번째 이유는 다형성에 기인한다. 객체지향 코드에서의 모든 호출(메시지 전달)은 절차형 언어에서 잠재적인 case문에 해당된다. 보통 수신자 클래스에 관해 모르기 때문에 어떤 호출 문장(메소드)은 비정형 언어 상태의 수많은 연산을 은폐시킬 수 있다. 원칙적으로 시스템 내에는 상당한 수의 클래스들이 존재한다. 일반적으로 case문은 McCabe 수를 빠르게 증가시키기 때문에 McCabe 척도를 사용할 경우 다형성을 다룰 방법을 결정하여야 하는데 지금까지의 적용에서 이 점은 거의 고려되지 않았다. 다른 하나는 연산 자체가 프로그램 순서를 갖기 때문에 연산의 복잡도를 측정해야 한다는 점이다. 우리가 다형성을 갖는 문장을 세지 않는다면 큰 McCabe 수를 갖는 연산이 존재하기란 아주 드물고 매우 특수한 경우를 제외하고는 절대로 10을 넘지 않게 될 것이다. 그러나 McCabe의 수는 OOSE에서 사용될 수 있다. 사용 케이스는 여러 객체를 하나의 특정한 순서로 연결시키면 되고 이러한 순서는 상당한 복잡도를 갖게 될 것이다. 하나의 사용 케이스에 대한 McCabe 수를 구함으로써 그 사용 케이스의 복잡도를 측정하게 된다. 여기에서 상호 작용 다이어그램(interaction diagram)이 척도를 계산하기 위한 도구로서 사용된다.

3.2 적용 실험

기존의 여러 척도들 가운데서 Halstead의 소프트웨어 과학(software science)[Hals 77]과 McCabe의 사이클로매틱 수(cyclomatic number)[McCa 76]가 객관적 관점에서 가장 우수하고 널리 사용되는 척도로 나타났다. 그러나 이 두 척도로 객체지향 프로그램의 복잡도를 측정하기에는 부적당함이 [More 90b]에서 실험적으로 증명되었다. Moreau의 연구팀은 객체지향 시스템에 대한 PEEM(Programming Environment Evaluation Methodology)의 기초를 세우기 위해 다음과 같은 실험을 하였다. 이들은 그래픽 편집기를 구성하는 각 모듈을 각각 C와 C++로 개발하여 이 모듈들에 Halstead와 McCabe의 척도를 적용한 결과 표 1과 표 2의 결과를 얻었다. 표 1은 같은 기능을 갖는 그래픽 에디터에 관련된 모듈을 여러 개발 그룹으로 나누어 C 와 C++ 언어로 개발한 시간을 비교한 것이다. 표 2는 개발된 각 모듈에 대해 Halstead와 McCabe가 제안한 척도를 적용한 결과를 보여 준다.

표 1에서 보는 것처럼 C++언어로 개발한 시간이 C언어로 개발한 시간의 65% 밖에 걸리지 않았음에도 불구하고 표 2에서 나타난 바와 같이 McCabe의 V(G)값과 Halstead의 척도 중 V값을 보면 C++ 언어로 개발한 모듈이 C 언어로 개발한 모듈보다 약 2배 정도 높게 나왔다. 그러나 위의 결과에서 상대적으로 짧은 개발 시간을 갖는 C++ 모듈에 척도를 적용한 결과 복잡도가 높게 나왔다고 하여 반드시 품질이 떨어진다고는 단정할 수 없다. 이는 두 언어의 각기 표현되는 언어의 전형이 다른 것이다. 따라서 McCabe의 V(G)값과 Halstead의 척도는 객체지향 프로그램에 잘 적용된다고 볼 수 없다.

이처럼 객체지향 소프트웨어의 복잡도 측정과 규모 산정등에 관한 이론적·직관적·경험적

표 1 C와 C++ 언어를 사용한 모듈 개발 시간

개발그룹 / 언어	S1	S2	S3	S4
C	2280초	2237초	1280초	2759초
C++	3204초	1432초	674초	1412초

표 2 각 모듈에 대한 기존 척도의 적용 결과

사용언어	C				C++			
개발그룹	S1	S2	S3	S4	S1	S2	S3	S4
V(G)	19	13	13	15	33	28	27	29
V	9206	9660	8991	8259	15695	15440	14365	15268

관점 제시 연구 및 객체지향 패러다임에 적합한 척도의 필요성을 보인 실험적 연구등을 통해 볼 때 객체지향 소프트웨어의 정량적 복잡도 측정을 위한 보다 정형화된 복잡도 측정 모델에 대한 연구가 절실함을 알 수 있다.

4. 연구 동향

4.1 국내·외 연구 현황

4.1.1 국외 연구

객체지향 패러다임 관련 복잡도에 대한 국외의 몇몇 초기 연구들은 기존 척도들의 단점을 보여주었고 특히 객체지향 개념에 적합한 새로운 복잡도 관점들을 제시하였다. 이러한 복잡도 관점들은 대부분 이론적이거나 연구자들의 직관적 또는 경험적 관점에서 제시되었다. 이론적이라기 보다는 실험적인 연구로서 몇 가지 제안이 Morris에 의해서 이루어졌다[Morr 89]. Morris는 그의 논문에서 객체지향 패러다임과 소프트웨어 개발에 대하여 분석한 후 클래스당 메소드 수 등 아홉 가지의 척도를 제안하고 제안 척도를 Booch와 Seidewitz가 제안한 각각의 객체지향 방법론의 단계 중 어느 단계에서 측정 가능한 지를 사상시켜보는 연구를 하였다. Pfleeger는 새로운 척도의 필요성을 제안하고 객체지향 개발을 위한 비용 산정 모델을 고안 및 테스트하기 위하여 객체와 메소드의 수를 사용하였으며[Pfle 89, Pfle 90], Moreau와 Dominick는 객체지향 그래픽 정보 시스템에 적합한 세 가지의 척도를 제안하였으나 공식적, 시험가능한 정의를 제시하지는 못하였다[More 87, More 89].

또한 Moreau와 Dominick는 [More 90a, More 90b]에서 객체지향 기법의 양적 측정을 위한 새로운 주요 척도를 고안하고자 프로그래밍 환경 평가 방법론에 대한 연구를 하였다. Chidamber와 Kemerer는 객체지향 설계의 규모와 복잡도에 영향을 주는 요소를 측정하기 위해 특별히 고안된 여섯 가지의 직관적·이론적 척도를 제안하였으나 실험적 증명은 하지 않았다[Chid 91, Chid 94]. 한편, Jenson과 Bartley는 객체, 연산, 인터페이스 등 세 가지의 매개변수를 활용하여 프로그래밍 노력도를 산정하기 위한 객체지향 모델을 제안하였으며[Jens 91], Laranjeira는 명세와 구현사이에 자연스러운 대응을 갖는 객체지향 기법의 이점을 바탕으로 소프트웨어 개발 주기의 초기 단계에서 객체지향 시스템의 규모를 보다 잘 예측할 수 있는 모델을 제안하기도 하였다[Lara 90]. 이 외에도 객체지향 관련 척도 연구로 [Biem 91, Buth 91, Lake 92, Li 93, Tega 92] 등이 있으며 객체 기반 언어인 Ada 관련 척도 연구로는 [Gann 86, Dame 92, Shat 88] 등이 있다.

4.1.2 국내 연구

객체지향 프로그램의 품질 또는 복잡도와 관련한 국내의 몇몇 연구들이 있다. 먼저 [한규정 90, 한규정 91]에서는 객체지향 프로그램의 품질을 정성적으로 분석하기 위한 기준을 제시하였는데, 모듈 강도(cohesion)를 측정하기 위한 기준으로서 무강도, 분리 가능한 강도, 비대표적 강도, 이상형 강도 등을 제안하였으며, 결합도(coupling) 측정 기준으로서 무결합도, 은폐된 결합도, 부분 결합도, 열린 결합도 등을 제안하였다. 그리고 [김은미 93, 문양선 92, 유철중 90, 유철중 92, Yoo 93, 유철중 94]등에서 객체지향 소프트웨어의 복잡도 측면에서의 특성 분석, 기존 척도의 적용 가능성 및 적용시의 문제점 분석, 객체지향 패러다임에 적용 가능한 새로운 척도의 필요성 도출과 시험적 척도 제안 등의 단계적 연구를 수행하였다. 특히, [유철중 94]에서는 객체지향 프로그램의 복잡도 변인들을 대상으로 주요 인자를 추출하여 이들의 특성을 반영하는 정량적 복잡도 측정 모델을 제안하였다. 또한 [이병복 93]에서 ATSN을 이용한 Ada 타스킹 실행 시간 복잡도 표현에 관한 연구도 수행하였다.

4.2 세부 연구내용

4.2.1 Moreau와 Dominick의 연구

Moreau와 Dominick는 그래픽 정보 시스템을 설계하고 구현할 때 전통적인 방법보다 객체지향 방법을 사용하는 것이 효과적임을 실험적으로 증명하고, 특히 [More 89]에서 객체지향 환경에서 프로그래밍의 복잡도에 영향을 주

는 세 가지 주요 요인을 다음과 같이 제시하였다.

1) 현 객체에 의해서 보내질 수 있는 유일한 메시지의 수(message vocabulary size)

2) 사용될 객체들의 특성을 결정하기 위해 알아야 하는 상속 메카니즘의 복잡도(inheritance complexity)

3) 현 객체가 응답해야 하는 메시지의 수(message domain size)

혼합 상속(compound inheritance)과 다중 레벨 상속(multilevel inheritance)은 구성 객체의 능력을 결정하는 과정을 복잡하게 만들며, 따라서 객체의 구성과 이해, 유지보수 등에 드는 노력에 영향을 준다. 위의 세 가지 척도는 객체지향 시스템을 평가하는데 필요한 포괄적인 소프트웨어 척도 리스트는 아니다. 예를 들어, 이들 척도는 목적 객체로 보내지는 메시지의 지연 바인딩(late binding)을 지원하는 전통적 시스템에 비해 객체지향 시스템이 갖는 증가된 오버헤드라 할 수 있는 실행 효율과의 관련 사항을 설명하지 못한다.

4.2.2 Winblad등의 연구

Winblad등은 그들의 저서[Winb 90]에서 객체지향 프로젝트를 효율적으로 관리하기 위해서는 새로운 척도가 필요함을 역설하였으며, 질이 좋은 설계와 프로그램을 생성하는데 영향을 주는 요인들을 다음과 같이 제시하였다.

1) 메소드 당 라인 수
2) 클래스 당 메소드의 수
3) 클래스 당 메소드 사용자 수
4) 클래스 계층 구조의 평균 깊이
5) 메소드 당 메시지 전송의 평균 횟수
6) 한 클래스가 수정된 이후의 경과 시간
7) 재사용 라인 수

4.2.3 Chidamber와 Kemerer의 연구

Chidamber와 Kemerer는 객체지향 설계의 규모와 복잡도에 영향을 주는 요소를 측정하기 위해 다음과 같은 여섯 가지의 이론적 척도를 제시하고 그 근거와 관점을 설명하였다[Chid 91, Chid 94]. 이러한 척도 고안은 여러 프로젝트에서 객체지향 설계를 사용했던 한 업체내의 소프트웨어 공학자팀과 협력하여 이루어졌다. 최근 이들의 이론적·직관적 연구결과가 몇몇 연구에서 인용되고 있는 점은 특기할만하다.

1) 클래스당 가중치를 가진 메소드의 수(Weighted Methods Per Class : WMC)

2) 상속 트리의 깊이(Depth of Inheritance Tree : DIT)

3) 자노드의 수(Number of Children : NOC)

4) 객체간의 결합도(Coupling Between Objects : CBO)

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

6) 메소드에서 응집성의 결핍도(Lack of COhesion in Methods : LCOM)

4.2.4 Jenson과 Bartley의 연구

Jenson과 Bartley는 객체지향 소프트웨어를 개발하기 위한 노력은 다음의 세 가지 요인에 의해서 영향을 받는다고 주장하였다[Jens 91].

1) 객체의 수
2) 연산의 수
3) 인터페이스의 수

이러한 성분들은 시스템에 대한 영어식 표현으로부터 추출이 가능하며, 따라서 원시 코드를 바탕으로 하는 전통적인 모델과는 달리 소프트웨어 개발의 초기 단계에서 유용한 경험적 산정 모델(empirical estimation model)을 개발하는 것이 가능하다고 하였다. 그들은 소프트웨어를 개발하기 위한 노력은 다음과 같이 객체, 연산, 인터페이스들의 수에 대한 함수로서 정의된다고 제안하였다.

Man hours = (Object, Operations, Interfaces)

4.2.5 Buth의 연구

Buth는 그의 프로젝트에서 객체지향 시스템을 여러 관점에서 수학적으로 모델화 하였다[Buth 91]. 예를들어, 객체지향 시스템을 클래스들간의 관련성을 포함한 상속 그래프 관점, 클래스들의 집합으로서의 관점, 유한한 일련의 알파벳 집합으로서의 관점 등으로 모델화하고 각 모델별로 복잡도 요인을 제안하였는데, 상속 그래프의 노드 수, 상속 그래프의 깊이와 너비, 특정한 노드의 부모노드 수와 자노드 수, 클래

스내의 메소드와 변수들의 수, 그리고 프로그램 라인 수와 주석 라인 수 및 공백 라인 수로 구분되어지는 전체 텍스트 라인 수 등이 그 예이다.

4.2.6 Dumke등의 연구

Dumke등은 그들의 연구[Dumk 92a, Dumk 92b]에서 객체지향 프로그래밍 언어의 특성은 메시지 관점에서 정적 클래스 구조(static class structure)와 동적 모듈 구조(dynamic module structure)로 분리할 수 있기 때문에 전통적인 척도와는 다른 척도가 필요하다고 하였다. 전통적인 척도는 클래스의 수, 메소드의 수, 변수의 수 등과 같은 객체지향 시스템의 일반적인 특성에 대해서 의미가 있으며, 어떤 메소드의 복잡도를 측정하기 위해서는 LOC, McCabe 척도 등과 같은 전통적인 척도를 사용할 수 있다고 하였다. 이들은 Smalltalk/V를 기반으로 한 Neumann의 논문[Neum 92]을 인용하여 시스템 수준, 클래스 수준, 그리고 메소드 수준에서 복잡도에 영향을 주는 요인들을 다음과 같이 제시하였다.

1) 시스템 수준의 요인
① 클래스의 수
② 계층 구조의 깊이
③ 계층 구조의 너비
④ 평균 서브클래스 수
⑤ 평균 메소드 수
⑥ 평균 변수의 수
2) 클래스 수준의 요인
① 클래스 메소드 수
② 인스턴스 메소드 수
③ 클래스 변수의 수
3) 메소드 수준의 요인
① 클래스당 라인 수(LOC)
② McCabe의 사이클로메틱 수

한편, 객체지향 언어 특성상 다른 척도들을 정의하는 것이 필요하다고 하였는데 그 내용은 다음과 같다.

1) 상속 복잡도
2) 통신 척도
3) 클래스 계층 구조의 깊이
4) 재사용 정도
5) 지식 분배(knowledge distribution)
6) 포괄성(generics) 척도
7) 다형성 척도

4.2.7 Jacobson등의 연구

Jacobson등은 그들의 저서[Jaco 92]에서 코드를 포함한 산출물에 관한 전통적인 척도는 객체지향 소프트웨어에 있어서도 어느 정도 사용될 수 있지만 가장 일반적인 척도인 LOC는 실제로 객체지향 소프트웨어를 측정하기에는 상당한 거리가 있다고 밝히고 있다. 또한 객체지향 소프트웨어에서는 작성한 코드가 적을수록 코드가 더 많이 재사용되었음을 의미하며 이는 항상 그러한 것은 아니지만 산출물의 품질을 훨씬 높이는 결과를 가져다 준다고 보고 있다. 다음 사항들은 Jacobson등에 의해 제시된 객체지향 소프트웨어에 보다 적합한 척도들로서 실제 코드와 관련된 척도들을 보여준다.

1) 상속 계층 구조의 갯수, 너비와 깊이
2) 특정한 연산을 사용하는 클래스의 갯수
3) 특정 클래스의 상위 클래스 갯수
4) 특정 클래스의 하위 클래스 갯수
5) 하나의 클래스 또는 연산을 직접 사용한 횟수(사용자의 수)

4.2.8 Li등의 연구

Li등은 Chidamber등이 제안한 다섯 종류의 척도에 상속에 의한 결합도, 메시지 전달에 의한 결합도, 데이타 타입 추상화에 의한 결합도 등 세 종류의 결합도 척도와 두 종류의 크기(size) 척도를 포함시킨 10개의 척도를 Classic-Ada로 작성된 2개의 상용 시스템(UIMS, QUES)에 적용하였다. 이들은 실험 방법으로서, 3년에 걸쳐 수집된 유지보수 노력 데이타와 10개의 척도 측정값들간의 회귀분석(regression analysis)을 실시하는 통계적 검증실험을 하였다[Li 93].

4.2.9 문양선의 연구

이 연구에서는 소프트웨어 공학 연구실의 석사과정에 있는 4명의 실험대상자가 여러 C++ 텍스트에 있는 프로그램들의 내용을 이해하는데 어려움을 주는 요인들을 실험을 바탕으로 직관적으로 제시하였다[문양선 92]. 제시된 요

인들은 프로그램을 이해하는데 어려움을 주는 요인들이다.

1) 클래스 수
2) 각 클래스에서 정의된 멤버함수의 수
3) 상속 계층 구조의 깊이
4) 멤버함수의 크기
5) 멤버함수의 재사용 정도
6) 멤버함수 사용시 발생하는 인터페이스 복잡도
7) 멤버함수 실행 경로

4.2.10 유철중의 연구

이 연구에서는 여러 연구에서 제시된 요인들을 중복성을 배제하되, 원시코드로부터 그 값을 얻어낼 수 있고 측정의 자동화가 가능한 다음과 같은 요인들을 추출하였다.

1) 클래스의 수
2) 메소드의 수
3) 상속 계층 구조의 깊이와 너비
4) 한 객체가 응답해야 할 서로 다른 메시지의 수
5) 한 객체가 보내는 서로 다른 메시지의 수
6) 메소드의 라인 수
7) 메소드의 재사용 정도
8) 다른 클래스의 인스턴스 변수 사용 수
9) 메소드 사용시 발생하는 인터페이스의 복잡도
10) 메소드 실행 경로

이러한 요인들의 값을 C++ 원시프로그램으로부터 측정하고, 그 값을 대상으로 인자분석(factor analysis)을 시행하였다. 그 결과, LOC 크기 및 캡슐화로 인한 인터페이스 관련 복잡도 인자와 클래스를 중심으로한 계층구조 관련 복잡도 인자 및 실행 경로에 따른 재사용 관련 복잡도 인자 등 세가지 대표 변인이 추출되었다. 이 연구에서는 이렇게 추출된 세가지 대표 변인들의 특성을 반영한 복잡도 측정 모델을 제안하였다. 즉, 제안된 모델은 객체지향 프로그램의 주요 메카니즘인 캡슐화와 상속성 및 재사용성을 반영한 정량적 척도로서, 프로그램 내의 각 클래스 단위로 구문적 성질을 측정하는 정적 복잡도 척도와 멤버 함수의 호출에 따른 인터페이스 복잡도 및 멤버함수의 재사용 정도를 고려하여 복잡도를 측정하는 동적 복잡도 척도로 나누어 제안하였다[유철중 94].

4.2.11 Tegarden등의 연구

Tegarden등[Tega 92]은 이상의 연구자들과는 다른 주장의 연구를 하였다. 즉, LOC나 Halstead의 소프트웨어 과학, McCabe의 사이클로매틱 복잡도 등과 같은 전통적 척도들이 객체지향 시스템의 복잡도 측정에도 유용함을 실험을 통하여 증명하였다. 이들은 하나의 간단한 회계 시스템을 네가지 객체지향 시스템으로 설계·구현하여 실험하였다. 즉, 비다형성(Without Polymorphism)과 비상속성(Without Inheritance)을 갖는 시스템, 다형성과 비상속성을 갖는 시스템, 비다형성과 상속성을 갖는 시스템, 다형성과 상속성을 모두 갖는 시스템 등 네가지로 설계·구현하여 위의 척도값들을 측정하였는데, 그 결과 전형적인 다형성과 상속성을 갖는 시스템이 가장 복잡도가 낮음을 보였다.

그러나 Tegarden등은 객체지향 시스템에 기존의 척도 적용시 여러 의문점이 있음을 지적하였다. 그들은 결론에서 상속에 따른 부작용이나 메시지 전달 등에 의한 복잡도 등 객체지향 시스템의 전형적 특성들의 측정에 별도의 척도가 필요함을 강하게 주장하였다.

5. 맺음말

객체지향 기법의 사용이 점차 일반화됨에 따라 현재의 기술 환경으로부터 객체지향 소프트웨어 개발 및 유지보수 환경으로 전환하는 경우 그 기대를 충족시키기 위해서는 프로세스와 산출물의 복잡도 측정을 위한 척도가 필요하다. 지금까지 살펴보았듯이 몇몇 연구에서 객체지향 시스템의 복잡도 측정을 위한 척도의 필요성 제기와 실제 척도 제안이 이루어졌다.

이들 여러 연구에서 제시된 복잡도 관련 요인들은 연구자들간에 중복성을 띤 요인들이 많이 있다. 이는 이러한 척도들이 객체지향 시스템의 복잡도 측정에 꼭 반영되어야 할 요인들임을 반증하며, 체계적인 실험과 검증을 거치면 객체지향 시스템에 적합한 모델 및 함축적인

정량적 복잡도 측정 모델을 얻을 수 있을 것으로 기대된다.

앞으로의 연구 과제로는 첫째, 보다 큰 규모의 프로그램과 다양한 용도의 프로그램에 방법론을 적용하여 실험해 보는 연구가 필요하다. 둘째, 전형적인 객체지향 소프트웨어 개발을 목표로 하는 프로젝트를 통해서 얻어진 산출물을 대상으로한 소프트에어 척도 연구가 필요하다. 셋째, 객체지향 프로그램의 기본적 복잡도 요인들의 값과 제안 모델의 값을 측정하기 위한 자동화 도구(metric analyzer)의 개발이 필요하다. 넷째, 정확한 복잡도 값 산출과 적용을 위해 인수 전달 및 함수의 호출등과 관련된 인터페이스 복잡도, 통신 복잡도 등을 반영하는 포괄적 개념의 객체지향 소프트웨어 복잡도 척도에 대한 지속적인 연구가 이루어져야 한다. 다섯째, 정량적 측정을 소프트웨어 수명 주기에 적절히 활용하는 방안에 대한 연구가 필요하다. 여섯째, 객체지향 프로그램의 데이타 구조 및 데이타 흐름에 기반을 둔 복잡도 측정 연구가 필요하다.

참고문헌

[1] J. M. Bieman, Deriving Measures of Software Reuse in Object Oriented Systems, Technical Report CS-91-112, Colorado State University, July 1991.

[2] B. J. Bowman and W. A. Newman, "Software Metrics as a Programming Training Tool," The Journal of Systems and Software, Vol. 13, No. 2, pp. 139-147, 1990.

[3] Angelika Buth, Software Metrics for Object-Oriented Programming Languages, (German) GMD Report 545, Birlinghoven, 1991.

[4] S. R. Chidamber and C. F. Kemerer, "Towards a Metrics Suite for Object Oriented Design," OOPSLA '91 Conference Proceedings, pp. 197-211, Oct. 1991.

[5] S. R. Chidamber and C. F. Kermerer, "A Metrics Suite for Object Oriented Design," IEEE Trans. on Software Engineering, Vol. 20, No. 6, pp. 476-493, 1994.

[6] S. Damerla and S. M. Shatz, "Software Complexity and Ada Rendezvous : Metrics Based on Nondeterminism," The Journal of Systems and Software, Vol. 17, No. 2, pp. 119-127, 1992.

[7] R. Dumke, Measurement Based Software Development, (German) Vieweg Publisher, Wiesbaden Braunschweig, 1992.

[8] R. Dumke, K. Neumann, and K. Stoeffler, "The Metirc Based Compiler-A Concurrent Requirement," ACM SIGPLAN Notices, Vol. 27, No. 12, pp. 29-38, Dec. 1992.

[9] J. D. Gannon, E. E. Katz, and V. R. Basili, "Metrics for Ada Packages : An Initial Study," Communications of the ACM, Vol. 29, No. 7, pp. 616-623. July 1986.

[10] M. H. Halstead, Elements of Software Science, New York : Elsevier North Holland, 1977.

[11] I. Jacobson, Object-Oriented Software Engineering, Addison-Wesley Publishing Company, Inc., 1992.

[12] R. L. Jenson and J. W. Bartley, "Parametric Estimation of Programming Effort : An Object-Oriented Model," The Journal of Systems and Software, Vol. 15, No. 2, pp. 107-114, May 1991.

[13] A. Lake and C. Cook, "A Software Complexity Metric for C++," Fourth Annual Workshop on Software Metrics, Oregon, March 22-24, 1992.

[14] L. A. Laranjeira, "Software Size Estimation of Object-Oriented Systems," IEEE Transactions on Software Engineering, Vol. 16, No. 5, pp. 510-522, May 1990.

[15] W. F. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability," The Journal of Systems and Software, Vol. 23, No. 2, pp. 111-122, 1993.

[16] T. J. McCabe, "A Complexity Measure," IEEE Transactions on Software Engineering, Vol. 2, No. 4, pp. 308-320, Dec. 1976.

[17] D. R. Moreau, A Programming Environment Evaluation Methodology for Object-Oriented Systems, Ph. D. Dissertation, University of South-western Louisiana,

Sept. 1987.

[18] D. R. Moreau and W. D. Dominick, "Object-Oriented Graphical Information Systems : Research Plan and Evaluation Metrics," The Journal of Systems and Software, Vol. 10, No. 1, pp. 23-28, 1989.

[19] D. R. Moreau and W. D. Dominick, "A Programming Environment Evaluation Methodology for Object-Oriented Systems : Part Ⅰ-The Methodology," Journal of Object-Oriented Programming, pp. 38-52, May/June 1990.

[20] D. R. Moreau and W. D. Dominick, "A Programming Environment Evaluation Methodology for Object-Oriented Systems : Part Ⅱ-Test Case Application," Journal of Object-Oriented Programming, pp. 23-32, Sept./Oct. 1990.

[21] K. Morris, Metrics for Object Oriented Software Development, Master's Thesis, M. I. T., Cambridge, M. A., 1989.

[22] K. Neumann, Implementation of Software Metrics in Smalltalk, (German) Degree Dissertation, TU Magdeburg, Jan. 1992.

[23] S. L. Pfleeger, "A Model of Cost and Productivity for Object Oriented Development," Contel Technology Center Technical Report, 1989.

[24] S. L. Pfleeger and J. D. Palmer, "Software Estimation for Object-Oriented Systems," Fall International Function Point Users Group Conference, San Antonio, Texas, Oct. 1-4, pp. 181-196, 1990.

[25] S. M. Shatz, "Towards Complexity Metrics for Ada Tasking," IEEE Transactions on Software Engineering, Vol. 14, No. 8, pp. 1122-1127, Aug. 1988.

[26] D. Taenzer, M. Ganti, and S. Podar, "Object-Oriented Software Reuse : The Yoyo Problem," Journal of Object-Oriented Programming, pp. 30-35, Sept./Oct. 1989.

[27] D. P. Tegarden, "Effectiveness of Traditional Software Metrics for Object-Oriented Systems," Proceedings of the 25th Hawaii International Conference on System Sciences, pp. 359-368, Jan. 1992.

[28] L. J. Waguespack, Jr. and S. Badlani, "Software Complexity Assessment : An Introduction and Annotated Bibliography," ACM SIGSOFT Software Engineering Notes, Vol. 12, No. 4, pp. 52-71, Oct. 1987.

[29] A. L. Winblad, S. D. Edwards, and D. R. King, Object-Oriented Software, Addion-Wesley Publishing Company, Inc., 1990.

[30] C. J. Yoo, Y. S. Kim, and O. B. Chang, "Quantitative Analysis of C++ Classes Complexity," Proceeding of Joint Conference on Software Engineering '93, Fukuoka, Japan, Nov. 17-19, pp. 245-252, 1993.

[31] H. Zuse, Software Complexity : Measures and Methods, Walter de Gruyter & Co., Berlin, 1990.

[32] 김은미, OOP 복잡도 척도 SOMEFOOT의 제안과 실험적 평가, 전북대학교, 석사학위논문, 1993. 8.

[33] 문양선, OOP의 복잡도에 영향을 주는 요인 분석에 관한 연구, 전북대학교, 석사학위논문, 1992. 2.

[34] 유철중, 이종득, 김용성, 장옥배, "객체 중심 프로그램의 소프트웨어 척도," '90 가을 학술 발표 논문집, 한국정보과학회, 제17권 제 2호, pp. 651-654, 1990.

[35] 유철중, 문양선, 김용성, 장옥배, "객체지향 언어와 Halstead의 소프트웨어 사이언스," '92 가을 학술 발표 논문집, 한국정보과학회, 제19권 제 2호, pp. 631-634, 1992.

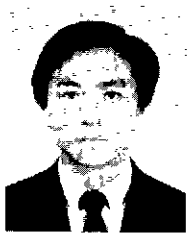
[36] 유철중, 객체지향 프로그램의 주요 메카니즘을 이용한 정량적 복잡도 측정 모델, 전북대학교 대학원, 박사학위논문, 1994. 8.

[37] 이병복, 유철중, 김용성, 장옥배, "ATSN을 이용한 Ada Tasking 실행시간 복잡도 표현에 관한 연구," 한국통신학회 논문지, 제18권 제5호, pp. 695-707, 1993. 5.

[38] 한규정, 김정아, 이경환, "추상자료형과 상속성을 갖는 C++ 클래스의 품질 평가 기준," 한국정보과학회 논문지, 제 17권 제 5호, pp. 550-559, 1990.

[39] 한규정, 김정아, 이경환, "객체지향 프로그램의 품질 평가 모델," '91 봄 학술 발표 논문집, 제 18권 제 1호, pp. 339-342, 1991.

유 철 중



1982 2 전북대학교 전산통계학과 졸업(이학사)
1985 8 전남대학교 대학원 계산통계학과 졸업(이학석사)
1994 8 전북대학교 대학원 전자계산학과 박사학위 취득(이학박사)
1982~1985 전북대학교 전자계산소 근무
1985~현재 기전여자전문대학 전자계산과 부교수
주요관심분야 : 소프트웨어공학, 객체지향 시스템, 인간과 컴퓨터 상호작용, 멀티미디어 시스템 등임

김 용 성

1978 2 고려대학교 수학과 졸업(이학사)
1991 2 광운대학교 대학원 전자계산학과 졸업(이학박사)
1985~1986 호서대학교 자연대학 전자계산학과 전임강사
1986~현재 전북대학교 자연과학대학 컴퓨터과학과 부교수
주요관심분야 : 지능형 교수 시스템(ITS), 소프트웨어공학, 컴퓨터알고리즘 등임

장 옥 배

1966 2 고려대학교 수학과 졸업(이학사)
1974~1980 죠지아 주립대, 오하이오 주립대 박사과정수료
1988 산타바바라대 졸업(이학박사)
1990~1991 영국 에딘버러대 객원 교수
1980~현재 전북대학교 자연과학대학 컴퓨터과학과 교수
주요관심분야 : 소프트웨어공학, 인공지능, CAI, 수치해석 등임