객체지향 소프트웨어의 설계 스타일 지침을 기반으로 하는 객체점수 척도

문 양 선[†] · 유 철 중[††] · 장 옥 배[†††]

요 약

객체지향 소프트웨어 척도는 객체들의 규모와 구성 및 객체들간의 관계를 고려하면서 전반적인 객체지향 특성을 반영하여야 한다. 본 논문에서는 객체지향 소프트웨어의 규모 및 복잡도에 영향을 주는 4가지의 객체지향 패러다임 관련 특성(요인)들을 정하고, 객체지향 설계 스타일 지침들을 기반으로 객체지향 소프트웨어의 품질 측정을 위한 객체점수 척도를 제안한다. 그 지침들은 기존의 지침들에 하나의 새로운 지침을 추가한 것으로서 인지 심리학에서의 청크 이론을 기반으로 제시된 것들이다. 제안된 객체점수 척도의 타당성 검증을 위해서 실험적 분석을 행하였는데, 이는 제안 척도가 전반적으로 객체지향 개념 및 특성을 잘 반영하는지를 평가하기 위한 것으로 같은 출력을 갖는 두 C++ 프로그램을 제안된 척도로 측정하여 비교하였다. 이러한 실험 결과 제안한 객체점수 척도의 유용성이 입증되었다.

Object-Point Metrics Based on Design Style Guidelines of Object-Oriented Software

Yang-Sun Moon[†] · Cheol-Jung Yoo[††] · Ok-Bae Chang[†††]

ABSTRACT

Object-oriented software metrics must reflect overall object-oriented characteristics including the size and structure of objects and the relation among them. This paper determines four object-oriented paradigm related features(factors) that affect the size and complexity of an object-oriented software and suggests object-point metrics for software quality measurement based on the object-oriented design style guidelines. The guidelines, added a new one to the existing guidelines, were proposed by applying the chunk theory of cognitive psychology. The suggested metrics were verified by experimental method. The experimental verification evaluates whether the suggested metrics reflected overall object-oriented concepts and characteristics. In this experiment, two C++ programs that output same result are compared by measuring with the suggested metrics. As the result, the usefulness of the suggested metrics was verified.

1. 서 론

품질 좋은 객체지향 소프트웨어의 개발을 돕기 위해 여러 연구에서 설계 및 프로그래밍 스타일 지침들이 제시되었다[1, 2, 3, 4]. 그러나 제시된 지침들 대부분이 객체지향의 특정 언어의 문법에 기반을 둔 프로그래밍 지침들이며 몇몇의 설계 지침들은 추상적이며 정성적인 성질(예; '불필요한 상속을 제거하라.' '추상화를 지향하라.' 등)을 띠고 있어 객체지향 소프트웨어 설계 단계에 이 지침들을 적용시키기는 어렵다. 객체지향 소프트웨어의 품질을 향상시키기 위해서는 소프

※ 이 논문은 1997년도 전북대학교의 지원 연구비에 의하여 연구되었음.

† 정 회 원 : 서해대학 사무자동화과 교수
†† 종신회원 : 전북대학교 컴퓨터과학과 교수, 영상·정보신기술 연구소 연구원
††† 정 회 원 : 전북대학교 컴퓨터과학과 교수, 영상·정보신기술 연구소 연구원

논문접수 : 1997년 11월 18일, 심사완료 : 1998년 8월 6일

트웨어의 설계가 잘 이루어져야 하므로 정성적인 설계 지침이 아니라 구체적이며 정량적인 지침들이 요구된다. 이러한 필요성에 따라 문헌 문양선, 유철중, 장옥배[5]는 인지심리 이론을 바탕으로 구체적이고 정량적인 객체지향 설계 스타일 지침을 제안하였다.

한편 객체지향 소프트웨어의 품질을 평가하기 위한 복잡도 척도도 여러 연구에서 제안하였다[6, 7, 8]. Moreau와 Dominick[6]은 메시지의 수 및 상속 레벨등을 척도로 제안하였고, Dumke[7]는 시스템 수준, 클래스 수준, 그리고 메소드 수준에서 복잡도에 영향을 주는 요인들을 제시하였다. Kemerer와 Chidamber[8]는 객체지향 설계의 규모와 복잡도에 영향을 주는 요소를 측정하기 위해 클래스당 가중치를 가진 메소드의 수(Weighted Methods Per Class; WMC), 상속 트리의 깊이(Depth of Inheritance Tree; DIT), 자노드의 수(Number of Children; NOC), 객체간의 결합도(Coupling Between Objects; CBO), 클래스에 대한 반응도(Response For a Class; RFC), 그리고 메소드에서 응집성의 결핍도(Lack of Cohesion in Methods; LCOM)를 척도로 제안하였다. 이상의 척도들의 가장 큰 단점은 소프트웨어 전체 복잡도를 하나의 숫자로 정량화하지 않은, 복잡도에 영향을 주는 요인들인 단순 척도(simple measures)들이기 때문에 사용자 입장에서 복잡도를 전체적으로 판단하기 어렵다는 것이다. 이러한 문제점을 해결하기 위해서 유철중, 김용성, 장옥배[9]는 문헌 [6], [7], [8]등 다른 연구에서 제안한 척도들을 복잡도에 영향을 주는 요인들을 중복성을 배제하여 선정한 후, C++ 프로그램을 대상으로 그 값들을 측정하여 그 결과를 대상으로 통계적 분석 방법인 인자 분석(factor analysis)을 행함으로써 주요 인자를 추출하고, 이러한 인자들을 반영하여 객체지향 프로그램의 전체 복잡도를 하나의 수치로 정량화하는 측정 모델을 제안하였다. 이 척도의 단점은 C++ 언어에 초점을 두었기 때문에 적용 범위가 좁다는 것이다. 김갑수, 신영길, 우치수[10]는 한 클래스 내에서 정보 흐름의 정도와 객체들 간의 정보 흐름의 정도를 표현하기 위하여 엔트로피 개념을 이용하여 복잡도를 하나의 숫자로 나타낼 수 있도록 척도를 제안하였는데, 측정이 객체간의 정보 흐름에 초점을 두었기 때문에 이 척도 또한 클래스나 메소드의 규모, 상속, 캡슐화 정도 등 객체지향 특성을 포괄적으로 반영하지 못한다는 단점이 있다.

본 논문에서는 기 제안된 객체지향 설계 스타일 지침에 몇몇 새로운 지침을 추가하여 이를 토대로 객체들의 규모, 구성, 그리고 객체들 간의 관계와 전반적인 객체지향 개념들을 반영하여 소프트웨어 전체의 규모나 복잡도 및 난이도를 하나의 숫자로 표현할 수 있는 객체점수(object-point) 척도를 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구에 대하여 논하고, 3장에서는 문양선, 유철중, 장옥배[5]의 연구에서 제안한 객체지향 설계 스타일 지침과 본 논문에서 추가한 새로운 몇몇 정의와 지침을 소개하고, 4장에서는 이러한 설계 스타일 지침을 기반으로 한 객체지향 소프트웨어의 객체점수 척도를 제안한다. 5장에서는 제안된 척도의 타당성을 검증하기 위한 실험을 하며, 마지막으로 6장에서는 본 논문의 결론과 향후 연구 방향을 논한다.

2. 관련 연구

Albrecht[11]는 소프트웨어 개발 비용산정 방법으로 기능 점수(function point) 모형을 제시하였다. 여기에서는 외부 입력, 외부 출력, 내부 논리 파일, 외부 연계용 파일, 외부 조회가 많을수록 개발비가 높아진다고 보고, 개발할 소프트웨어의 기능 점수를 계산하기 위해 소프트웨어의 기능을 증대시키는 요인 별로 해당 숫자를 추정하여 기입한 후 프로젝트의 특성에 적절한 가중치를 선택하여 곱함으로써 각 요인 별 기능 점수를 계산한 후, 이들을 합하여 총 기능 점수를 산출하도록 하고 있다. 즉, 총 기능 점수란 소프트웨어의 개발 규모, 복잡도, 난이도 등을 하나의 숫자로 집약시키는 것이다. 이 모형은 그 유용성과 간편성 때문에 현재까지 나와 있는 비용 산정방법으로는 최선이라는 평판을 받고 있지만[12], 객체지향의 특성을 고려하지 않았기 때문에 객체지향 소프트웨어에 적용시키기는 어렵다.

Sneed[13]는 기존의 LOC(Lines Of Code)와 기능 점수(function point) 모형[11]이 객체지향 소프트웨어의 규모 예측을 위한 좋은 척도가 되지 못함을 지적하고 객체지향 소프트웨어의 개발 비용 예측을 위한 새로운 척도로서 객체 점수(object point)를 제안하였다. 제안된 객체 점수는 클래스 포인트(class point), 메시지 포인트(message point), 프로세스 포인트(process point)로 계산된다.

클래스 포인트는 객체의 형, 데이터 속성의 수(Att-

ribute), 관계의 수(Relation), 메소드나 함수들의 수(Method), 변화의 정도(Change_Rate)로 측정하며, 그 식은 다음과 같다.

Class_Point = {(Attribute) + (Relation × 2) + (Method × 3)} × Change_Rate

메시지 포인트는 클래스의 통합과 통합 시험 노력을 측정하는 기반이 된다. 메시지는 그것의 형과 복잡도 정도(Complexity_Rate), 인수의 수(Parameter), 메시지 송신 클래스들의 수(Source), 메시지 수신 클래스들의 수(Target), 변화의 정도(Change_Rate)로 측정하며 다음과 같이 계산된다.

Message_Point = {Parameter + (Source × 2) + (Target × 2) × Complexity_Rate} × Change_Rate

프로세스 포인트는 시스템 테스팅 노력을 측정하는 식으로서, 그것의 형(Process_Type)과 복잡도(Complexity_Rate) 및 다양성(Variation)으로 측정하며, 그 식은 다음과 같다.

Process_Point = {Process_Type + Variation) × Complexity_Rate

이 연구에서는 4가지 형태의 프로세스를 고려하여 각 프로세스에게 다음과 같은 가중치를 주고 있다. 일괄 프로세스는 2, 실시간 프로세스와 온라인 프로세스는 4, 시스템 프로세스는 6을 취하고, 또한 복잡한 프로세스는 1.25를, 단순한 프로세스는 0.75를 곱한다. 최종적으로 객체 점수는 다음과 같이 이 세 포인트의 합으로 계산된다.

Object_Point = Class Points + Message Points + Process Points

이 척도의 단점은 클래스 포인트 계산에서 메소드의 규모와 상속의 문제를 고려하지 않아 상속된 속성들과 메소드들은 한 번만 계산되고 그것이 선언된 베이스 클래스나 슈퍼클래스만 고려하고 있으며, 메시지 포인트 계산에서는 메시지 전달 복잡도, 즉 메시지 전달 경로를 고려하지 않고 있으므로 메시지 송·수신 클래스의 수만 같으면 메시지 전달 과정이 아무리 복잡하다 하더라도 메시지 포인트의 점수가 같게 측정된다. 또한 프로세스 포인트 계산에서는 'Complex_Rate'를 고려하기 위한 정확한 기준이 설정되어 있지 않다.

정태인[14]등은 Sneed가 제안한 객체 점수가 객체지향 소프트웨어를 사용자 관점에서 나타내기 힘들다고 지적하고, Sneed 연구의 메시지 포인트에서 말하는 관계(Relation)에 대하여 새롭게 정의한 후, 객체 점수(object point)를 다음과 같이 단순화시켰다.

Class points = Attribute + Method
Message points = Relations + Parameters
Object oriented points = Class points + Message points

그러나 이 척도 또한 메소드의 규모와 메시지 전달 경로 등을 고려하지 않고 있다.

클래스의 규모는 메소드의 규모에 영향을 받고 객체간의 통신 복잡도는 메시지 전달 복잡도에 영향을 받으므로 객체지향 소프트웨어 규모와 복잡도를 측정하기 위한 척도는 반드시 메소드 규모와 메시지 전달 경로를 고려해야 한다. 따라서 본 논문에서는 이 모두를 고려한 객체점수(Object-Point) 척도를 제안하고자 한다.

3. 객체지향 설계 스타일 지침

문양선, 유철중, 장옥배[5]의 연구에서는 인지심리에서 인간의 단기 기억(short-term memory)의 한계를 나타내는 '7±2 chunks' 이론[15]과 인지 실험을 통해 얻은 '내포 구조의 한계 수 3'으로부터 8가지의 객체지향 설계 스타일 지침을 유도하였는데, 본 논문에서는 몇 가지 정의를 추가하고 이를 바탕으로 하나의 지침을 추가하였다.

3.1 '7±2 chunks' 이론으로부터 유도된 지침

'7±2 chunks' 이론은 Miller[15]가 인간의 단기 기억의 기억 폭(memory span)에 대한 한계를 알아보기 위한 실험을 통해 얻어진 것으로서 낱자, 단어, 의미가 있는 구절(phrase) 등과 같은 일련의 나열된 정보(청크)를 인간의 단기 기억에 저장할 수 있는 항목의 수가 7±2개 라는 이론이다. 인지심리학에서 청크는 '한 입력 문자열을 하나의 구조화된 그룹으로 재배열한 것'이라고 정의된다[15]. 예를 들어, 하나의 단어는 음소(phonemes)들의 청크이고, 한 문장은 단어들의 청크이다. 문헌 [5]에서는 이러한 인지심리에서의 청크 개념을 프로그램의 구성 성분들(연산자나 피연산자, 프로그램 코드, 블록, 파일)과 관련시켜 프로그램에서의 청크들을 유도하였으며, 특히 객체지향 프로그램 성분들(인스턴스 변수, 메소드, 클래스, 상속 계층구조)에 청크

개념을 적용시켜 다음과 같은 정의를 하였다.

정의 1. 하나의 **인스턴스 변수**(instance variable)는 하나의 청크이다.
정의 2. 관련있는 문장으로 이루어진 하나의 **메소드**는 하나의 청크이다.
정의 3. 메소드와 인스턴스 변수들로 이루어진 하나의 **클래스**는 하나의 청크이다.
정의 4. 클래스들로 이루어진 하나의 **상속 계층구조** (class inheritance hierarchy)는 하나의 청크이다.

다음 정의는 본 논문에서 추가한 것으로 관련 있는 단어들로 이루어진 숙어 또한 하나의청크가 될 수 있다는 개념으로부터 정의되었다.
정의 5. 직접적인 참조관계에 있는 클래스들의 모임 또한 하나의 청크이다.

여기에서 직접적인 참조관계에 있는 클래스들의 모임이란 **한 클래스와 그것의 직접적인 서브클래스들의 모임**에 속하나 **상속과는 무관하면서 직접적인 참조 관계를 갖고 있는 클래스들의 모임**을 말한다. 즉, 전자는 'generalization' 또는 'aggregation' 관계를 갖는 상속구조와 관련된 클래스들을 말하며, 후자는 상속구조에는 포함되지 않으나 메시지 송·수신과 관계가 있는 클래스들의 모임을 의미한다.

위의 객체지향 프로그램에서의 청크에 대한 정의와 '7±2 chunks' 이론을 통해서 다음과 같은 7가지의 객체지향 설계 스타일 지침을 유도하였다[5].

지침 1. 하나의 메소드 내의 문장 수는 7±2개 이하가 바람직하다.
지침 2. 하나의 클래스내의 메소드 수는 7±2개 이하가 바람직하다.
지침 3. 하나의 클래스 내의 인스턴스 변수 수는 7±2개 이하가 바람직하다.
지침 4. 하나의 상속 계층구조내의 전체 클래스 수는 7±2개 이하가 바람직하다.
지침 5. 하나의 프로그램 내의 상속 계층구조의 수는 7±2개 이하가 바람직하다.
지침 6. 한 클래스의 직접적인 자녀클래스 수는 7±2개 이하가 바람직하다.
지침 7. 한 클래스와 직접적인 참조 관계를 갖는 클래스 수는 7±2개 이하가 바람직하다.

지침 7에서 직접적인 참조 관계란 상속 관계가 아닌 메시지 송·수신 관계를 의미한다. 문헌 [5]에서는 지침 1을 '메소드 내의 라인 수'로 규정지었으나 본 논문에서는 문장(statement) 수로 정정하였다. 그 이유는 똑 같은 내용의 프로그램 코드라도 코딩을 어떻게 하느냐에 따라 라인 수가 달라질 수 있으므로 메소드의 규모를 측정하기 위해서는 라인 수보다는 문장 수가 더 타당하기 때문이다. 또한 문헌 [5]에서 지침 6을 '상속 계층구조의 너비(width)는 7±2개 이하가 바람직하다'로 표현하였으나 본 논문에서는 클래스 중심의 표현으로 변경하였으며, 아울러 지침 7을 새롭게 추가하였다. 이 두 지침은 앞에서 정의한 정의 5를 기반으로 제안된 것이다.

3.2 인간의 내포구조 처리의 한계에 대한 실험으로부터 유도된 지침

정의 1~5에서 정의한 청크들은 프로그램의 단순 평면구조(flat structure)를 갖는 구성 성분들에 관한 것들이다. 따라서 내포 블록(nested block)처럼 하나의 청크를 이해하는데 다른 청크의 이해가 선행되어야 하는 성분들, 즉 프로그램의 이해 측면에서 내포구조의 성질을 갖는 성분들에 대한 청크의 정의가 별도로 필요하다. 객체지향 프로그램의 이해 측면에서 내포구조의 성질을 띠는 성분들은 '서브클래스-슈퍼클래스', '호출 메소드-피호출 메소드' 이다. 그 이유는 서브클래스와 호출 메소드를 이해하기 위해서는 슈퍼클래스와 피호출 메소드의 이해가 선행되어야 하기 때문이다. 따라서 객체지향 프로그램에서 내포구조를 갖는 청크에 대해서 다음과 같이 정의한다.

정의 6. 호출 메소드와 그에 대응하는 **피호출 메소드 세트**(set)는 하나의 청크이다.
정의 7. 슈퍼클래스와 그것의 **서브클래스 세트**는 하나의 청크이다.

정의 6은 피호출 메소드가 다시 다른 메소드를 호출한다면 호출된 메소드도 같은 청크에 포함됨을 뜻하며, 정의 7 또한 서브클래스가 또 자신의 서브클래스를 갖는다면 그 서브클래스까지도 같은 청크에 포함됨을 뜻한다.

문헌 [5]에서는 이러한 내포 구조에 대한 인간의 인지적 특성을 알아보기 위하여 실험을 하였다. 그 실험

에서 인간의 다차원 자극에 대한 정보처리 능력의 한계는 3차원이라는 결과를 얻어 이를 '**내포구조의 한계 수 3**(Limit Number 3 of Nested Structure)'이라고 정의하였다.

객체지향 프로그램에서 3차원적 내포구조의 성질을 갖는 것은 상속 계층구조의 깊이가 3인 경우, 메소드의 호출 길이가 3인 경우이므로 위의 정의 6과 7, 그리고 '내포구조의 한계 수 3'을 적용하여 다음과 같은 지침을 제안하였다[5].

지침 8. 클래스 상속 계층구조의 깊이(depth)는 3이하가 바람직하다.
지침 9. 메소드의 호출 경로 길이(path length)는 3이하가 바람직하다.

지금까지 제안된 9가지의 지침에 대하여 구체적인 의미와 지침 적용의 적합성, 그 지침을 준수하기 위한 방안 등에 관한 토론은 문헌 [5]에 자세히 기술되어 있다.

4. 객체점수

4.1 정의

본 논문의 객체 점수 모형은 기능 점수 모형의 응용으로서, 객체지향 소프트웨어의 규모와 복잡도 및 난이도 등에 영향을 주는 객체지향 패러다임 관련 특성(요인)들의 품질을 정량적으로 평가한 값을 '**객체점수**(*Object-Point*)'라 정의한다. 따라서 객체점수를 계산하는 식의 형태는 기존의 기능 점수 모형과 유사하나 식에서 사용된 매개변수와 매개변수에 적용되는 가중치는 다르다.

4.2 제안 척도

객체지향 소프트웨어의 규모, 복잡도, 난이도 등에 영향을 주는 요인은 메소드, 클래스, 상속 계층구조, 메시지 등이므로 이러한 요인들의 품질 척도를 다음과 같이 제안한다.

4.2.1 메소드 척도

메소드의 규모와 복잡성은 메소드 내의 문장 수에 의해 영향을 받으므로 메소드 품질 척도를 (식 1)과 같이 제안한다.

$$MTM = \sum_{i=1}^{a} S\_No(M_i) \times W_i \qquad \text{(식 1)}$$

여기서 *MTM(MeThod Metrics)*은 메소드 척도를 뜻하고, a는 한 프로그램 내에 정의되어 있는 메소드의 수이며, $S_No(M_i)$는 i번째 메소드 M_i의 문장 수이다. 그리고 W_i는 $S_No(M_i)$의 가중치이다. *MTM*은 문헌 [5]의 지침에 관한 의미에서 밝힌 바와 같이 코드 재사용 정도를 측정한다.

4.2.2 클래스 척도

클래스의 규모와 복잡성은 클래스들의 메소드 수와 인스턴스 변수 수에 의해 영향을 받으므로 클래스 척도를 (식 2)와 같이 제안한다.

$$CLM = \sum_{i=1}^{b} M\_No(C_i) \times W_{i1} + IV\_No(C_i) \times W_{i2} \qquad \text{(식 2)}$$

여기에서 *CLM(CLass Metrics)*은 클래스 척도를 뜻하고, b는 한 프로그램 내에 정의되어 있는 클래스의 수, $M_No(C_i)$와 $IV_No(C_i)$는 각각 i번째 클래스 C_i의 메소드 수와 인스턴스 변수 수이다. 또한 W_{i1}과 W_{i2}는 각각 $M_No(C_i)$와 $IV_No(C_i)$에 대한 가중치이다. *CLM*은 문헌 [5]의 지침에 관한 의미에서 밝힌 바와 같이 응집성과 캡슐화 정도를 측정한다.

4.2.3 상속 계층구조 척도

프로그램 내에 정의되어 있는 상속 계층구조의 복잡도는 프로그램내의 상속 계층구조 수, 각 상속 계층구조 내에 정의되어 있는 클래스 수, 그리고 각 상속 계층구조의 깊이에 의해 영향을 받는다. 따라서 상속 계층구조 척도를 (식 3)과 같이 제안한다.

$$IHM = \sum_{i=1}^{c} C\_No(H_i) \times W_i + \sum_{j=1}^{d} Depth(C_j) \times W_j + H\_No \times W$$

(식 3)

여기에서 *IHM(Inheritance Hierarchy Metrics)*은 상속 계층구조 척도를 뜻하고, c와 d는 각각 한 프로그램 내에 정의되어 있는 상속 계층구조의 수와 클래스 수이며, $C_No(H_i)$는 i번째 상속 계층구조 내의 클래스 수를, $Depth(C_j)$는 하나의 상속 계층구조 내에서 j번째 클래스의 상속 깊이를 나타내며, H_No는 프로

그램 내의 상속 계층구조 수를 나타낸다. W_i, W_j, W는 각각 *C_No(Hi)*, *Depth(Cj)*, 그리고 *H_No*에 대한 가중치이다. *IHM*은 문헌 [5]의 지침에 관한 의미에서 밝힌 바와 같이 추상화 정도를 측정한다.

4.2.4 메시지 척도

메시지의 복잡성은 한 프로그램 내에 정의되어 있는 모든 클래스들에 대해서 각 클래스의 서브클래스 수와 참조 관계를 갖는 클래스 수, 그리고 각 클래스들에 정의되어 있는 메소드들의 호출 길이에 의해 영향을 받는다. 따라서 메시지 척도를 (식 4)와 같이 제안한다.

$$MSM = \sum_{i=1}^{e}(Ch\_No(C_i)\times W_{i1} + RC\_No(C_i)\times W_{i2}) + \sum_{j=1}^{f} Length(M_j)\times W_j \quad \text{(식 4)}$$

여기에서 *MSM(MeSsage Metrics)*은 메시지 척도를 뜻하고, *e*와 *f*는 각각 한 프로그램 내에 정의되어 있는 클래스 수와 메소드 수를 뜻한다. 그리고 $Ch_No(C_i)$와 $RC_No(C_i)$는 각각 *i*번째 클래스 C_i의 서브클래스 수와 상속과는 무관하면서 클래스 C_i와 참조관계에 있는 클래스 수이다. 또한 W_{i1}과 W_{i2}는 각각 $Ch_No(C_i)$와 $RC_No(C_i)$의 가중치이고, $Length(M_j)$는 *j*번째 메소드 M_j의 호출 길이이며, W_j는 $Length(M_j)$의 가중치이다. *MSM*은 객체간 결합성과 인터페이스 복잡도 정도를 측정한다.

4.2.5 객체점수

객체점수는 메소드 척도, 클래스 척도, 상속 계층구조 척도, 그리고 메시지 척도 값을 합하여 계산한다. 이에 대한 식은 (식 5)와 같다.

$$Object\text{-}Point = MTM + CLM + IHM + MSM \quad \text{(식 5)}$$

MTM, *CLM*, *IHM*, 그리고 *MSM*은 객체지향의 서로 다른 특성을 측정하지만, 4가지의 척도 값을 합한 *Object-Point*는 프로그램이 객체지향 특성들을 전반적으로 잘 이용하였는가와 성분들의 규모와 구성 및 그것들 간의 관계가 어느 정도 복잡한지를 나타낸다. (식 1)~(식 4)의 4가지 척도에서 나타나는 각각의 가중치들은 이 객체점수를 구하기 위한 중요한 인수들이다. 이러한 가중치들을 구하기 위해서 또 다른 인지실험을 하였는데 이에 관해서는 다음 절에서 자세히 기술한다.

4.3 가중치 부여 실험 및 가중치

4.3.1 부여 기준

소프트웨어의 규모, 복잡도, 난이도 등에 영향을 주는 요인들을 바탕으로 한 객체점수를 계산하기 위해서는 각 요인들의 수에 따라 가중치를 다르게 부여하는 기준의 설정이 필요하다.

<표 1>은 객체지향 소프트웨어의 품질에 영향을 주는 요인들과 그 요인들의 척도 값이 반영하는 객체지향 특성, 그 요인들의 품질에 영향을 주는 변수들, 그리고 그에 대한 가중치 부여 기준 및 가중치를 보여준다. 인간의 단기기억은 한정된 정보량을 어떤 활성화 상태로 유지시키는 용량을 말하는데, 그 용량은 '7 ±2 chunks'임이 밝혀졌으므로 <표 1>에서 가중치 부여 기준은 3장에서 소개한 지침들을 토대로 하였다. 프로그램의 단순 평면구조와 관련된 요인들에는 3단계, 즉 '적다', '보통', '많다'로 구분하였으며, 내포 구조와 관련된 요인들에는 4단계로 구분하였다(<표 1>의 가중치에 대한 설명은 4.3.2~4.3.4절 참조).

4.3.2 가중치 부여를 위한 실험

가중치 부여를 위한 실험 데이터들은 각 청크들을 이해하는데 요구되는 노력도를 측정할 수 있도록 선정되어야 한다. 따라서 각 척도에서 사용된 가중치를 부여하기 위한 가장 적절한 실험 데이터로는 본 논문 3장에서 소개한 설계 지침들을 잘 반영하고 있는 구조를 갖는 프로그램들이어야 하나 기존에 개발된 프로그램에서는 찾아보기 힘들고, 찾은 경우라 할지라도 각 구성 성분(청크)들간의 난이도가 다를 수 있어 이상적인 실험이 되지 못하기 때문에, 본 논문에서는 인지이론에서의 청크(관련 있는 단어가 이루는 문장 청크, 문장들로 이루어지는 절 청크 등) 개념을 반영할 수 있는 데이터를 선정하였다. 이를 위해서 본 논문에서는 '공동 번역 성서'에 쓰여 있는 문장들을 각 청크 개념과 부합되도록 추출하였다. '공동 번역 성서'를 선택한 이유는 데이터 선정의 편리성 때문이다. 왜냐하면 성서는 다양한 내용을 포함하고 있으며 내용 별 분류가

〈표 1〉 가중치 부여 기준 및 가중치
〈Table 1〉 Weighting criteria and weighted values

복잡도 요인	요인들의 척도 값이 반영하는 객체지향 특성	요인들의 품질에 영향을 주는 변수	가중치 부여 기준	가중치 (실험결과)
메소드	코드 재사용 정도	메소드 내의 코드 수	적다(5미만)	92
			보통(5~9)	172
			많다(10이상)	230
클래스	응집성/캡슐화 정도	클래스 내의 메소드 수	적다(5미만)	365
			보통(5~9)	891
			많다(10이상)	1287
		클래스 내의 인스턴스 변수 수	적다(5미만)	10
			보통(5~9)	64
			많다(10이상)	161
상속 계층구조	추상화 정도	클래스 상속 계층구조 내의 클래스 수	적다(5미만)	2035
			보통(5~9)	4579
			많다(10이상)	6614
		프로그램 내의 클래스 상속 계층구조 수	적다(5미만)	10581
			보통(5~9)	23808
			많다(10이상)	34390
		클래스 상속 계층구조의 깊이	얕다(1)	2544
			보통(2)	4664
			깊다(3)	9582
			매우 깊다 (4이상)	19928
메시지	결합성/인터페이스 복잡도 정도	한 클래스의 자녀 클래스 수	적다(5미만)	2035
			보통(5~9)	4579
			많다(10이상)	6614
		한 클래스와 참조 관계를 갖는 클래스 수	적다(5미만)	2035
			보통(5~9)	4579
			많다(10이상)	6614
		메소드 호출 깊이	짧다(1)	495
			보통(2)	908
			길다(3)	1865
			매우 길다 (4이상)	3878

체계적이고, 각 내용마다 제목이 부여되어 있어 실험 후 결과 분석이 용이하기 때문이다. 특히 공동 번역 성서는 다른 성서보다 쉽게 풀이되어 있어 성경을 전혀 읽지 않은 사람들도 이해하기 쉬운 장점이 있다.

실험 대상자로는 성경을 읽은 경험이 없는 대학생 40명이 선정되었으며, 실험방법은 적절한 실험 데이터와 데이터에 대한 몇 가지 질문이 적혀있는 설문지를 학생들에게 나누어주어 그들이 각 질문에 완벽한 답을 할 수 있을 때까지의 시간(초)을 기록하게 하는 것이다. <표 1>에서의 가중치는 본 논문에서 행한 가중치 부여를 위한 실험을 통해서 얻어진 결과이다.

4가지 척도에서 사용된 가중치 부여를 위해 선정된 실험 데이터들은 크게 모두 4종류의 유형으로 구성된다. 그것들에 대하여 기술하면 다음과 같다.

(1) 데이터 유형 1

데이터 유형 1은 한 메소드 내의 문장 수가 적을 때(4미만), 보통일 때(5~9), 그리고 많을 때(10이상), 그 메소드를 이해하는데 요구되는 노력도를 알아보기 위한 실험 데이터이다.

프로그램에서의 청크 개념에서 설명한 것처럼 메소드는 서로 관련 있는 프로그램 문장들로 이루어진 청크로 볼 수 있으므로 본 논문에서는 메소드의 문장 수가 4일 경우, 9일 경우, 13일 경우 이해하는데 요구되는 노력도를 알아보기 위해 성서의 서로 다른 책에서 4절(데이터 유형 1-1), 9절(데이터 유형 1-2), 그리고 13절(데이터 유형 1-3)을 데이터로 선정하였다(예를 들면, '사무엘상 19:1~4', '열왕기하 6:24~32', 그리고 '예레미야 14:1~13' 등). 성서의 서로 다른 책에서 발췌한 이유는 이 실험이 내용면에서 전혀 관련이 없는 메소드에 대한 것이기 때문이다. 성서에서 1절은 실제 한 문장이 넘을 수도 있고 넘지 않을 수도 있으나 한 프로그램에서의 한 문장처럼 하나의 내용을 담고 있기 때문에 1절을 1문장으로 취급하였다.

(2) 데이터 유형 2

데이터 유형 2는 한 클래스 내의 메소드가 9개이고 각 메소드 내의 문장 수가 보통일 때(5~9개) 그 클래스를 이해하는데 요구되는 노력도를 알아보기 위한 실험 데이터이다. 이를 위해 성서를 구성하고 있는 책들 중 한 책에서만 9절씩 9구역을 선정하였다.

(3) 데이터 유형 3

데이터 유형 3은 클래스 내의 인스턴스 변수 수가 적을 경우(5미만), 보통일 경우(5~9), 많을 경우(10이상) 등 각각 그 클래스 내의 인스턴스 변수들을 인지하는데 요구되는 노력도를 알아보기 위한 실험 데이터이다. 이를 위해서 보통 명사 4개(데이터 유형 3-1), 9개(데이터 유형 3-2), 13개(데이터 유형 3-3)를 선정하였다.

(4) 데이터 유형 4

데이터 유형 4는 프로그램 성분의 내포구조의 깊이가 각각 1, 2, 3, 4인 경우, 그 성분을 이해하는데 요구되는 노력도의 비율을 측정하기 위한 실험 데이터이다. 즉, 1차원, 2차원, 3차원, 4차원의 정보를 처리하는데 걸리는 시간의 비율을 측정하기 위한 것이다. 선정된 데이터들은 상속 깊이 1의 경우에 대해서는 보통명사 2개씩 순서쌍을 이룬 데이터 항목 9개(데이터 유형 4-1), 상속 깊이 2의 경우에 대해서는 보통명사 3개씩 순서쌍을 이룬 데이터 항목 9개(데이터 유형 4-2), 상속 깊이가 3의 경우에 대해서는 보통명사 4개씩 순서쌍을 이룬 데이터 항목 9개(데이터 유형 4-3), 상속 깊이가 4의 경우에 대해서는 보통명사 5개씩 순서쌍을 이룬 데이터 항목 9개(데이터 유형 4-4)를 각각 선정하였다.

이상과 같이 주어진 데이터들에 대한 질문 내용은 데이터 유형 1과 2에 대해서는 학생들에게 데이터 안에 나타나는 사람의 이름과 주제를 완전히 파악하여 기록하고, 그렇게 하는데 걸리는 시간(초)을 측정하여 기록하도록 하였다. 그 이유는 사람의 이름을 기억하는 것은 프로그램내의 인스턴스 변수를 기억하는 것과 유사하며, 주제를 파악하는 것은 프로그램 내의 메소드나 클래스의 기능을 파악하는 것과 유사하기 때문이다. 데이터 유형 3과 4에 대해서는 주어진 데이터 유형을 완전히 암기하는데 걸리는 시간(초)을 기록하게 하였다. 데이터 유형 3에 대한 그 이유는 한 클래스 내에 정의된 인스턴스 변수는 클래스를 구현할 때, 즉 메소드를 정의할 때 이용되는데 그러한 메소드를 이해하기 위해서는 클래스에서 정의한 모든 인스턴스 변수를 기억하고 있어야 하기 때문이다. 데이터 유형 4에 대한 실험이 필요한 이유는 앞에서도 기술한 바와 같이 데이터 유형 4가 1차원, 2차원, 3차원, 4차원의 정보를 처리하는데 걸리는 시간의 비율을 측정하기 위한 데이터이기 때문이다. 이러한 실험을 통해서 얻어진 가중치 부여 기준에 따른 가중치는 앞의 <표 1>에서 보여주고 있다.

4.3.3 실험 결과

<표 2>는 4.2절의 제안 척도 식에서 나타나는 가중치 인수 값을 얻어내기 위해 실험한 데이터 유형별 평균 결과를 보여준다.

〈표 2〉 가중치 부여를 위한 실험 결과
〈Table 2〉 Experimental results for weighting

데이터	데이터 유형 1			데이터 유형 2	데이터 유형 3			데이터 유형 4			
	1-1	1-2	1-3		3-1	3-2	3-3	4-1	4-2	4-3	4-4
평균 (초)	92	172	230	921	10	64	161	190	352	724	1505
비율						1		3.0	5.5	11.3	23.5

<표 2>에서 비율 1, 3.0, 5.5, 11.3, 23.5는 데이터 유형 3-2에 대한 4-1, 4-2, 4-3, 그리고 4-4의 상대적 비율이다. 이 비율은 프로그램 성분의 내포구조의 깊이가 0, 1, 2, 3, 4일 때 그 성분을 파악하는데 요구되는 노력도의 비율을 나타낸다.

4.3.4 가중치 부여

4가지 척도에서 사용된 가중치를 부여하기 위해서 <표 2>의 실험 결과를 직접·간접적으로 이용하며 구체적인 부여 내용은 다음과 같다.

(1) 메소드 내의 문장 수에 대한 가중치

<표 2>에서 데이터 유형 1-1, 1-2, 1-3의 실험 결과가 각각 문장의 수가 4, 9, 13인 경우 피실험자들이 그 문장들을 이해하는데 걸린 시간을 나타내므로, 이에 대한 가중치 부여는 그 평균 결과들을 그대로 적용한다. 즉, 메소드 내의 문장 수가 적을 경우(5미만)에는 92를, 보통일 경우(5~9)에는 172를, 많을 경우(10이상)에는 230을 취한다.

(2) 클래스 내의 메소드 수에 대한 가중치

한 클래스가 하나의 청크로 다루어지는 개념은 하나의 메소드를 청크로 다루는 개념의 확장개념이므로 이에 대한 가중치는 데이터 유형 1과 데이터 유형 2에 대한 실험 결과를 분석하여 적용한다.

앞에서도 설명한 것처럼 데이터 유형 1-2는 9개의 문장으로 이루어진 하나의 메소드를 이해하는데 걸리는 시간을 측정하기 위해 선정된 데이터로서 실험결과는 <표 2>에서 알 수 있는 것처럼 172초이었다. 따라서 만일 9개의 문장으로 이루어진 절들을 9개 제시하였다면 약 1548(≈172×9)초가 나왔어야 한다. 그러나

데이터 유형 2의(9 문장으로 이루어진 절을 9개 선정하여 제시한 데이터 임)의 실험 결과는 평균 921초이었다. 그 이유는 데이터 유형 2는 성서의 같은 책에서 발췌한 절들로서 그 내용들이 관련이 있어 서로 다른 책들에서 발췌하였을 경우보다 이해가 용이하였기 때문이다. 이 결과는 같은 클래스 내의 메소드들을 이해하는데 걸리는 시간이 서로 다른 클래스에 정의된 메소드들을 이해하는데 걸리는 시간보다 적게 걸릴 것이라는 직관적 사고를 잘 반영하고 있다. 이 두 경우의 비율(1548 : 921)은 약 1 : 0.6 정도이다. 따라서 클래스내의 메소드 수에 대한 가중치는 데이터 유형 1-1, 1-2, 1-3의 실험결과들의 평균값 (92+172+230)/3(=165)와 0.6 비율을 이용한다. 평균값을 이용하는 이유는 한 클래스 내의 메소드들의 문장 수가 '적다', '보통', '많다'의 범위 중 어느 하나에 해당될 수 있기 때문이다. 따라서 클래스 내의 메소드 수가 4일 경우의 가중치는 165×4×0.6(=365)을, 9일 경우의 가중치는 165×9×0.6(=891)을, 그리고 13이상일 경우의 가중치는 165×13×0.6(=1287)으로 하여 이 세 값을 각각 클래스 내의 메소드 수가 적을 경우(5미만), 보통일 경우(5~9), 많을 경우(10이상)에 적용시킨다.

(3) 클래스 내의 인스턴스 변수 수에 대한 가중치

<표 2>에서 데이터 유형 3-1, 3-2, 3-3의 실험 결과가 클래스 내의 인스턴스 변수 수가 4일 때, 9일 때, 13일 때에 관한 실험 결과이므로 이에 대한 가중치 부여는 그 평균 결과들을 그대로 적용한다. 즉, 한 클래스 내의 인스턴스 변수 수가 적을 경우(5미만)에는 10을, 보통일 경우(5~9)에는 64를, 많을 경우(10이상)에는 161을 가중치로 취한다.

(4) 상속 계층구조 내의 클래스 수에 대한 가중치

하나의 상속 계층이 하나의 청크로 다루어지는 개념은 한 클래스가 하나의 청크라는 개념의 확장 개념이므로 여기에서도 (2)의 방법과 같은 방법으로 가중치를 구한다. 즉, 클래스 내의 메소드 수에 대한 가중치들의 평균 값 (365+891+1287)/3(=848)과 0.6 비율을 이용한다. 따라서 상속 계층구조 내의 클래스 수가 4일 경우의 가중치는 848×4×0.6(=2035)을, 9일 경우의 가중치는 848×9×0.6(=4579)을, 그리고 13이상일 경우의 가중치는 848×13×0.6(=6614)으로 하여 이 세 값을 각각 상속 계층구조 내의 클래스 수가 적을 경우(5미만), 보통일 경우(5~9), 많을 경우(10이상)에 적용시킨다.

(5) 프로그램 내의 상속 계층구조 수에 대한 가중치

한 프로그램이 하나의 청크로 다루어지는 개념은 하나의 상속 계층구조가 하나의 청크라는 개념의 확장 개념이므로 (2) 및 (4)와 같은 방법으로 가중치를 구한다. 즉, 상속 계층구조내의 클래스 수에 대한 가중치들의 평균 값 (2035+4579+6614)/3(=4409)와 0.6 비율을 이용한다. 따라서 프로그램 내의 상속 계층구조 수가 4일 경우의 가중치는 4409×4×0.6(=10581)을, 9일 경우의 가중치는 4409×9×0.6(=23808)을, 그리고 13이상일 경우의 가중치는 4409×13×0.6(=34390)으로 하여 이 세 값을 각각 프로그램 내의 상속 계층구조 수가 적을 경우(5미만), 보통일 경우(5~9), 많을 경우(10이상)에 적용시킨다.

(6) 클래스 상속 구조의 깊이에 대한 가중치

상속 깊이에 따라 하나의 클래스를 이해하는데 요구되는 노력도는 다르다. 따라서 상속 깊이가 각각 1, 2, 3, 그리고 4인 클래스별 가중치는 (2)에서 계산한 클래스에 대한 가중치들의 평균 848과 데이터 유형 3-2, 4-1, 4-2, 4-3 그리고 4-4의 비율인 1 : 3.0 : 5.5 : 11.3 : 23.5를 이용한다. 따라서 상속 깊이가 0인 클래스의 가중치는 848, 1인 클래스의 가중치는 848×3(=2544), 2인 클래스는 848×5.5(=4664), 3인 클래스는 848×11.3(=9582), 그리고 4이상인 클래스는 848×23.5(=19928)를 적용시킨다.

(7) 한 클래스의 자녀클래스 수에 대한 가중치

이에 대한 가중치는 상속 계층구조 내의 클래스 수에 대한 가중치를 그대로 적용하기로 한다. 그 이유는 클래스라는 관점이 같기 때문이다. 즉, 서브클래스 수가 5미만 경우는 2035, 5~9일 경우에는 4579, 10이상인 경우는 6614를 취한다.

(8) 한 클래스와 참조 관계를 갖는 클래스 수에 대한 가중치

이에 대한 가중치 역시 상속 구조 내의 클래스 수에 대한 가중치를 그대로 적용하기로 한다. 그 이유 또한 클래스라는 관점이 같기 때문이다. 즉, 서브클래스 수가 5미만인 경우는 2035, 5~9일 경우에는 4579, 10이상인 경우는 6614를 취한다.

(9) 메소드 호출 길이에 대한 가중치

호출 길이에 따라 하나의 메소드를 이해하는데 요구되는 노력도는 다르다. 따라서 호출 길이가 각각 1, 2, 3, 그리고 4인 메소드 별 가중치는 (1)에서 계산한 메소드에 대한 가중치들의 평균 (92+172+230)/3(=165)와 데이터 유형 3-2, 4-1, 4-2, 4-3, 그리고 4-4의 비율인 1 : 3.0 : 5.5 : 11.3 : 23.5를 이용한다. 호출 길이가 0인 메소드의 가중치는 165, 1인 메소드는 165×3(=495), 2인 메소드는 165×5.5(=908), 3인 메소드는 165×11.3(=1865), 그리고 4이상인 메소드는 165×23.5(=3878)를 적용시킨다.

이상의 가중치들은 실험 대상자의 수나 그들의 지능에 따라 달라질 수가 있어 부여된 가중치가 절대적일 수는 없으나, 청크 개념 및 객체점수 계산 방법의 특성상 데이터 유형 별 실험 결과의 비율을 고려했기 때문에 실험 대상자들이 달라진다고 해도 본 실험에서 부여된 가중치의 범위를 크게 벗어나지 않을 것이다.

기능 점수 모형[11]의 문제점 중 하나는 제안 모델의 가중치가 주관적이라는 점이다[16]. 본 논문에서도 가중치 부여를 위한 실험 데이터 선정이나 가중치 계산 방법에 있어서 약간은 객관성이 결여되어 있어 가중치 부여에 대한 더 많은 연구가 이루어져야 한다. 그러나 본 논문의 척도가 프로그램을 이해하는데 인지심리의 복잡도를 측정하는 것이므로 인지심리를 바탕으로 한 실험 데이터 선정과 가중치 설정은 나름대로 의의를 갖고 있다.

5. 척도 적용실험 및 결과 분석

제안한 객체점수 척도의 유용성을 검증하기 위한 실험적 방법으로서 같은 결과를 갖지만 다른 구조를 갖는 [부 록]의 프로그램 P1과 P2 프로그램에 대해 각각 객체점수를 측정하는 과정 및 결과를 보이고자 한다. P1 프로그램은 보다 정련된 C++ 코딩 방법에 대하여 논하고 있는 Traister의 저서[17]로부터 선정된 프로그램으로서 'main' 함수를 제외한 부분만을 보여주고 있다. <표 3>은 P1을 분석한 결과이다.

P1은 세 클래스 'Alpha', 'Beta', 그리고 'Chi'가 모두 상속 관계를 갖고 있다. 따라서 클래스 'Beta'는 상속 깊이가 1이며 클래스 'Chi'는 2이다. 'Alpha'와 'Beta'는 서브클래스 수가 모두 1이며 상속과 무관하면서 참조 관계에 있는 클래스는 존재하지 않는다. 또한 클래스 'Beta'의 생성자 'Beta()'와 멤버함수 'display()'가 실행될 때 클래스 'Alpha'의 생성자 'Alpha()'와 멤버함수 'display()'가 호출되어 실행되며, 클래스 'Chi'의 생성자 'Chi()'와 멤버함수 'display()'가 실행될 때 클래스 'Beta'의 생성자 'Beta()'와 멤버함수 'display()'가 호출되어 실행된다. 따라서 클래스 'Alpha'의 생성자 'Alpha()'와 멤버함수 'display()'의 최대 호출길이는 모두 2이며, 클래스 'Beta'의 생성자 'Beta()'와 멤버함수 'display()'의 호출길이는 1이 된다.

〈표 3〉 P1의 성분 분석
〈Table 3〉 Components analysis of P1

클래스 명	상속 깊이	자녀 클래스 수	참조 클래스 수	인스턴스 변수 수	멤버함수명	호출 길이	문장 수
Alpha	0	1	0	2	Alpha()	2	2
					display()	2	1
					two_times()	0	1
Beta	1	1	0	2	Beta()	1	2
					display()	1	2
Chi	2	0	0	2	Chi()	0	2
					display()	0	2

P1의 객체점수는 다음과 같다.

$$Object\text{-}Point(P1) = MTM + CLM + IHM + MSM$$
$$= (2\times92+1\times92+1\times92+2\times92+2\times92+2\times92+2\times92) + (3\times365+2\times365+2\times365+2\times10+2\times10+2\times10) + (3\times2035+1\times2544+2\times4664+1\times10581) + (1\times2035+1\times2035+0+2\times908+2\times908+1\times495+1\times495)$$
$$= 1104+2615+28558+8692$$
$$= 40965$$

P2는 P1의 클래스 상속 구조를 비상속 구조로 재구성한 프로그램이다. 따라서 P2는 객체지향 특성이 완전히 무시된 프로그램으로서 코드 재사용률과 클래스 응집성이 낮고 클래스 간 메시지 전송은 존재하지 않는다. <표 4>는 P2의 성분들을 분석한 결과이다. P2는 세 클래스 'Alpha', 'Beta', 그리고 'Chi'가 상속 관계를

갖고 있지 않기 때문에 독립적인 클래스 3개로 구성되어 객체지향 프로그램이 아닌 객체기반(object-based) 프로그램이며(이 논문에서는 독립적인 클래스 하나도 하나의 상속 계층 구조로 취급), P1보다 인스턴스 변수와 메소드 내의 문장 수가 많은 반면 메소드 호출 길이는 0이다.

〈표 4〉 P2의 성분 분석
〈Table 4〉 Components analysis of P2

클래스 명	상속 깊이	자녀 클래스 수	참조 클래스 수	인스턴스 변수 수	멤버함수명	호출 길이	문장 수
Alpha	0	0	0	2	Alpha()	0	2
					display()	0	1
					two_times()	0	1
Beta	0	0	0	4	Beta()	0	4
					display()	0	1
Chi	0	0	0	6	Chi()	0	6
					display()	0	1

P2의 객체점수는 다음과 같다.

$$Object\text{-}Point(P2) = MTM + CLM + IHM + MSM$$
$$= (2\times92+1\times92+1\times92+4\times92+1\times92+6\times172+1\times92) + (3\times365+2\times365+2\times365+2\times10+4\times10+6\times64) + (1\times2035+1\times2035+1\times2035+3\times10581+0) + (0+0+0)$$
$$= 1952+2999+37848+0$$
$$= 42799$$

<표 5>는 P1과 P2 프로그램에 대해 본 논문에서 제안한 4가지 척도로 측정하여 비교한 표이다. 위에서 설명한 것처럼 P2는 상속이 깨짐으로 인해 P1에 비해 코드 재사용률과 응집성은 낮아진 반면, 클래스들간의 메시지 전송이 없어 결합성이나 인터페이스 복잡성은 전혀 없다. 측정결과 P1의 객체점수 값이 P2의 객체점수 값보다 적게 나타났는데, 이는 P2가 P1보다 객체지향 특성(특히 상속)을 효율적으로 이용하지 못하여 프로그램 성분들(메소드나 클래스)의 규모가 크고 비효율적으로 구성되어 있음을 뜻한다.

이러한 두 프로그램의 비교를 통해서 볼 때, 본 논문에서 제안한 척도는 메소드의 품질, 클래스의 품질, 그리고 클래스간의 인터페이스 복잡도 뿐만 아니라 클래스의 응집성 및 클래스들 간의 결합성, 그리고 캡슐화와 추상화 정도를 잘 반영하고 있다는 사실을 입증할 수 있다.

〈표 5〉 P1과 P2에 대한 4가지 척도 값의 비교
〈Table 5〉 Comparison on 4 metrics values for P1 and P2

척도	측정 값		반영된 객체지향 특성
	P1	P2	
MTM	1104	1952	코드 재사용 정도
CLM	2615	2999	응집성/캡슐화 정도
IHM	28558	37848	추상화 정도
MSM	86927	0	결합성/인터페이스 복잡성 정도

제안 척도의 좀 더 확실한 실험적 검증을 위해서는 실질적으로 같은 출력을 갖는 여러 개의 객체지향 프로그램들에 대하여 위와 같은 실험적 비교 평가가 필요하나 실제 환경에서 그러한 프로그램들을 구하기가 어렵다는 제약이 있다. 한편, 객체점수 척도 값을 자동 측정하여 주는 도구를 이용하면 실험에 편리를 기할 수 있을 것이다.

6. 결 론

본 논문에서는 객체지향 설계 지침들을 토대로 객체지향 소프트웨어의 규모 및 복잡도에 영향을 주는 메소드, 클래스, 상속 계층구조, 그리고 메시지의 품질을 측정하기 위한 4가지의 척도와 전체 프로그램의 품질을 측정하기 위한 객체점수 척도를 제안하였다. 척도를 제안하는데 토대가 되는 설계 지침들은 기존의 제안된 지침들에 새로운 지침을 추가한 것으로써 인지심리학에서의 청크 이론을 기반으로 제시된 것이다. 제안된 척도는 기존의 절차지향 소프트웨어의 개발비용 예측 모델 중 하나인 기능 점수 모형을 객체지향 소프트웨어의 특성에 맞도록 응용한 것으로서, 기존의 연구가 객체지향 소프트웨어의 개발비용 예측에 초점을 둔 모형인데 반해 본 논문에서의 척도는 복잡도 측정에 초점을 두고 제안되었다.

제안된 척도의 타당성 검증을 위해서 실험적 검증을 행하였는데, 이는 제안 객체점수 척도가 전반적으

로 객체지향 개념 및 특성을 잘 반영하는지를 평가하기 위한 것으로 같은 출력을 갖는 두 C++ 프로그램을 측정하여 비교하여 보았다. 이러한 실험을 통해서 제안 척도의 유용성이 입증되었다.

향후 연구로 객체점수 계산에 있어 부여된 가중치가 보다 객관적이 되도록 실험을 보강하는 것이 필요함은 물론, 제안 척도를 여러 실용 프로그램들에 적용해 보는 보완 연구 및 값을 자동적으로 측정하여 주는 객체점수 척도 분석기(object-point metrics analyzer)의 설계 및 구현이 필요하다.

참 고 문 헌

[1] T. Cargill, *C++ Programming Style*, Addison-Wesley Publishing Company, Inc., 1992.

[2] J. Coplien, *Advanced C++ Programming Styles and Idioms*, Addison-Wesley Publishing Company, Inc., 1992.

[3] C. Horstmann, *Mastering Object-Oriented Design in C++*, John Wiley & Sons, Inc., 1995.

[4] S. Meyer, *More Effective C++*, Addison-Wesley Professional Computing Series, 1996.

[5] 문양선, 유철중, 장옥배, "인지심리 이론을 반영한 객체지향 설계 및 프로그래밍 스타일 지침", 한국정보과학회 논문지(B), 제25권 제3호, pp.530-542, 1998.

[6] 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.

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

[8] C. F. Kemerer and S. R. Chidamber, "A Metrics Suite for Object-Oriented Design," *IEEE Transactions on Software Engineering*, Vol.20, No.6, June 1994.

[9] 유철중, 김용성, 장옥배, "객체지향 프로그램의 복잡도 측정을 위한 척도", 정보과학회 논문지, 제21권 제11호, pp.2039-2049, 1994.

[10] 김갑수, 신영길, 우치수, "엔트로피를 이용한 객체지향 프로그램의 복잡도 척도", 정보과학회 논문지(B), 제22권, 제12호, pp.1656-1666, 1995.

[11] A. J. Albrecht, "Measuring Application Development Productivity," *Proceedings of IBM Application Development Symposium*, Monterey, 1979.

[12] 이주헌, 실용소프트웨어공학론, 법영사, 1994.

[13] Harry M. Sneed, "Estimating the Development Costs of Object-Oriented Software," *European Software Control and Metric Conference*, May 1996.

[14] 정태인, 김수동, 류성렬, "객체지향 메트릭 분석을 통한 객체지향 점수 메트릭의 연구", 한국정보과학회 '96 학술발표논문집, pp.1499-1502, 1996.

[15] G. A. Miller, "The Magical Number Seven Plus or Minus Two :Some Limits on Our Capacity to Process Information," *Psychological Reviews*, Vol.63, pp.81-87, 1956.

[16] C. F. Kemerer and B. S. Porter, "Improving of the Function Point Measurement: An Empirical Study," *IEEE Transactions on Software Engineering*, Vol.18, No.11, pp.1011-1024, Nov. 1992.

[17] R. J. Traister, *Clean Coding in Borland C++*, M&T Publishing, Inc., 1994.

[18] E. J. Weyuker, "Evaluating Software Complexity Measures," *IEEE Transactions on Software Engineering*, Vol.14, No.9, pp.1357-1365, Sep. 1988.

[부 록]

상속 깊이가 2인 C++ 프로그램 예제(P1)

```
#include <stdio.h>
class Alpha {
   private:
   int x, y;
   public:
     Alpha(int i, int j)
      {  x = i;
         y = j; }
     void display()
      { printf("%d %d\n", x, y); }
     void two_times()
      { printf("%d %d\n", x*2, y*2); } };
class Beta : public Alpha {
   private:
      int a, b;
   public:
```

```
    Beta(int i, int j, int q, int r)
    : Alpha(i, j)
    { a = q;
      b = r; }
    void display()
    { Alpha::display();
      printf("%d %d\n", a, b); } };
class Chi : public Beta {
  private:
    double v, w;
  public:
    Chi(int i, int j, int q, int r, double x, double y)
    : Beta(i, j, q, r)
    { v = x;
      w = y; }
    void display()
    { Beta::display();
      printf("%f %f\n", v, w); } };
```

P1을 비상속구조로 재구성한 프로그램 예제(P2)

```
#include <stdio.h>
class Alpha {
  private:
  int x, y;
  public:
    Alpha(int i, int j)
    {  x = i;
       y = j; }
    void display()
    { printf("%d %d\n", x, y); }
    void two_times()
    { printf("%d %d\n", x*2, y*2); } };
class Beta {
  private:
    int x, y, a, b;
  public:
    Beta(int i, int j, int q, int r)
    { x = i;
      y = j;
      a = q;
      b = r; }
    void display()
    { printf("%d %d %d %d\n", x, y, a, b); } };
class Chi {
  private:
   int x, y, a, b;
   double v, w;
  public:
    Chi(int i, int j, int q, int r, double c, double d)
    { x = i;
      y = j;
      a = q;
      b = r;
      v = c;
      w = d; }
    void display()
    { printf("%d %d %d %d %f %f\n", x, y, a, b, v,
      w); } };
```

문 양 선

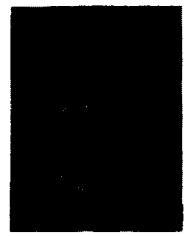
1986년 2월 전북대학교 전산통계학과 졸업(이학사)

1992년 2월 전북대학교 대학원 전산통계학과 졸업(이학석사)

1996년 전북대학교 대학원 전산통계학과 박사과정 수료

1995년~현재 서해대학 사무자동화과 조교수

관심분야 : 객체지향 소프트웨어 테스팅, 소프트웨어 품질평가 등



유 철 중

1982년 2월 전북대학교 전산통계학과 졸업(이학사)

1985년 8월 전남대학교 대학원 계산통계학과 졸업(이학석사)

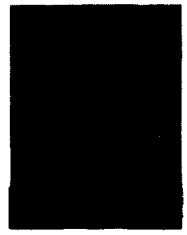
1994년 8월 전북대학교 대학원 전자계산학과 졸업(이학박사)

1982년 9월~1985년 3월 전북대학교 전자계산소 조교

1985년 4월~1996년 12월 전주기전여자대학 전자계산과 부교수

1997년 1월~현재 전북대학교 자연과학대학 컴퓨터과학과 전임강사

관심분야 : OOSE, Multimedia, HCI, Distributed Objects Computing Cognitive Science 등



장 옥 배

1966년 고려대학교 수학과 졸업(학사)

1974년~1980년 죠지아주립대, 오하이오주립대 박사과정 수료

1990년~1991년 영국 에딘버러대학교 객원교수

1980~현재 전북대학교 컴퓨터과학과 교수

관심분야 : 소프트웨어공학, 전산교육, 수치해석, 인공지능 등