메트릭을 이용한 객체 지향 설계 재구조화
(Restructuring of Object-Oriented Designs using Metrics)

이 병 정 † 우 치 수 ††
(Byung-Jeong Lee) (Chi-Su Wu)

요 약 객체 지향 설계를 재구조화하기 위해서는 메소드와 속성들 사이의 관계를 파악해야 한다. 메소드와 속성들이 동일한 클래스에 속하는지를 추론하는 과정은 클래스 자료 참조 분석과 메소드 호출 분석에 의존한다. 최근의 많은 소프트웨어들은 규모가 방대하고 복잡하여 개발자가 도구를 사용하지 않고 수작업으로 설계를 재구조화하기 어려우며 또한 너무 많은 시간과 노력이 요구된다. 본 논문에서는 메트릭을 이용하여 객체 지향 설계를 자동적으로 재구조화하는 방법을 기술한다. 먼저 메소드, 속성, 클래스, 그리고 그들의 관계를 추상화된 모델로 표시한다. 이 모델을 기반으로 객체 지향 설계를 정량적으로 측정할 응집도와 결합도 메트릭을 정의한다. 본 논문의 메트릭은 재구조화 과정 동안에 여러 다른 설계들을 효율적으로 비교하기 위한 유용한 기준을 제공한다. 기본 재구조화 행위들과 그 의미(semantics)를 정의하고 메트릭과 행위들을 사용하여 설계를 자동적으로 재구조화한다.

Abstract To restructure object-oriented designs, relationships between methods and attributes in classes must be identified. It depends on an analysis of references to attributes and method call structures to deduce which methods and attributes may be members of the same class. Recent commercial systems are too large and too complex to restructure their design by hand without tools and cost too much time and effort. This paper describes a restructuring approach using metrics to change object-oriented designs automatically. Abstract models are used to indicate methods, attributes, classes, and their connections. Cohesion and coupling metrics, based on the models, are defined to measure object-oriented designs quantitatively. The metrics provides useful criteria for comparing different designs efficiently during the restructuring. We define a set of primitive restructuring operations and its semantics, and restructure the designs with the metrics and the operations automatically.

1. 서 론

재구조화(restructuring)는 이해하기 쉽도록 또는 변경할 때 오류를 범하지 않도록 소프트웨어를 수정하는 과정이다 [1]. 코딩 스타일의 개선, 문서 편집, 프로그램 구성 요소의 변환, 또는 프로그램 모듈화 등이 모두 재구조화의 예로 볼 수 있다. 객체 지향 시스템에서 설계 구조가 좋지 않으면 유지보수 비용이 증가하고, 재사용이 어렵고, 그리고 소프트웨어 생명 주기가 짧아진다. 반면에 설계가 좋으면 명세, 설계, 코드의 검사, 수정, 그리고 재작성에 대한 필요를 줄일 수 있으므로 시간과 비용을 절약할 수 있고 제품의 품질을 높일 수 있다.

객체 지향 설계 재구조화는 설계 구조를 모듈화하는 과정으로서 이를 위해서는 메소드와 속성들 사이의 관계를 파악하고 기능을 추론해야 한다. 메소드와 속성들이 동일한 클래스에 속하는지를 추론하는 과정은 클래스 자료 참조 분석과 메소드 호출 분석에 의존한다. 지금까지 이러한 모듈화 과정은 대개 수작업으로 코드를 검사하고 편집하는 것에 의해 수행되었다. 그러나 최근의 많은 소프트웨어들은 규모가 방대하고 복잡하여 개발자가 도구를 사용하지 않고 수작업으로 설계를 재구조화하려면 너무 많은 시간과 노력이 요구된다. 그래서 자동적으로 수행되는 정밀한 설계 재구조화 방법이 필요하다. 이를 위해서는 초기 설계를 추상화 모델로 표시하고 이 모델을 메트릭을 사용하여 정확하게 정량화하여 측정해야 한다. 그리고 이 측정 결과를 기반으로 자동화된 재구조화 과정을 수행해야 품질이 우수한 설계를 얻을 수 있다.

† 학생회원 : 서울대학교 컴퓨터공학부
bjlee@selab.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
wuchisu@selab.snu.ac.kr
논문접수 : 2000년 10월 2일
심사완료 : 2001년 4월 30일

본 논문에서는 메트릭을 이용하여 객체 지향 설계를 재구조화하는 기법을 제안한다. 먼저 시스템을 추상화된 모델로 표시하고, 이 모델을 기반으로 응집도(cohesion)와 결합도(coupling) 메트릭을 정의한다. 응집도는 클래스에 속한 메소드와 속성들 사이의 관련성(relatedness)을 의미하고, 결합도는 클래스들 사이의 연결성(connectedness)을 의미한다. 추상화 모델을 기반으로 재구조화 행위 집합과 그 의미(semantics)를 정의하고 응집도와 결합도를 이용한 유전자 알고리즘(genetic algorithm)을 적용하여 객체 지향 설계를 재구조화한다.

본 논문의 구성은 다음과 같다. 2장에서 메트릭과 재구조화에 관한 기존 연구를 기술한다. 3장에서 전체적인 객체 지향 설계 재구조화 과정을 요약하고, 재구조화를 위한 모델과 결합도와 응집도 메트릭을 정의한다. Briand가 제안한 메트릭 성질을 이용하여 정의한 메트릭을 검증한다. 그리고 4장에서 재구조화 행위의 의미와 유전자 알고리즘을 이용한 재구조화 과정을 설명한다. 마지막으로 5장에서 결론과 향후 연구 계획을 기술한다.

2. 관련 연구

본 장에서는 응집도와 결합도를 포함한 메트릭에 대한 연구와 메트릭을 이용한 재구조화 기법에 대한 연구를 기술한다.

2.1 기존의 메트릭

객체 지향 시스템의 설계 품질을 평가하는 연구는 많이 수행되었다. Chidamber and Kemerer[2]는 메소드가 다른 클래스의 메소드나 인스턴스를 사용하는 경우 결합도(CBO)가 있다고 정의하였다. 이 방법은 한 클래스와 연관있는 다른 클래스 개수로 클래스 결합도를 간단히 측정할 수 있지만, 클래스 내에서 다른 클래스의 메소드나 인스턴스를 여러 번 사용할 때도 한 번 사용한 것과 같은 값을 가진다. 응집도 메트릭은 메소드의 유사성을 기반으로 서로 소(disjoint)인 클래스 속성 집합의 개수로 초기 LCOM(Lack of Cohesion in Methods)을 정의하였고, 이후에 [8]에서 공통 속성을 갖고 있지 않은 메소드 쌍의 개수에서 공통 속성을 갖고 있는 메소드 쌍의 개수를 뺀 값으로 LCOM을 다시 재정의하였다.

Li and Henry[3]는 결합도를 상속성, 메시지 전달, 그리고 추상 자료형의 세 가지 측면으로 나누어 정의하였다. 상속성 결합도는 [2]의 DIT와 NOC를 사용하였고, 메시지 전달에 의한 결합도(MPC)는 클래스에 정의된 호출 문장의 개수로 정의하였다. 그리고 추상 자료 형에 의한 결합도(DAC)는 클래스에 정의된 추상 자료 형의 개수로 정의하였다.

Hitz and Montazeri[9]는 메소드 그래프에서 연결된 그래프 요소들의 개수로 LCOM을 정의하였다. 메소드 사이에 호출이 있거나 또는 같은 속성을 접근하는 경우에 그래프의 두 메소드는 서로 연결된다. 또한 LCOM=1인 경우들을 세밀히 구분하기 위하여 연결성(connectivity) $C = 2*\frac{|E|-(|V|-1)}{(|V|-1)*(|V|-2)}$를 정의하였다. 결합도는 객체 수준 결합도(object-level coupling)와 클래스 수준 결합도(class-level coupling)로 구분하여 정의하였다. 객체 수준 결합도는 실행 시간 동안 객체 사이의 동적인 의존성에 기인하는 결합도를 의미하는데, 예를 들면 naive 객체가 아닌 객체의 자료 또는 메소드를 사용하는 관계를 객체 수준 결합도라고 한다. naive 객체가 되는 경우는 다른 객체의 서브객체가 되거나, 다른 메소드의 지역 변수가 되거나, 또는 부모 클래스로부터 상속받은 다른 객체의 서브객체가 되는 경우이다. 클래스 수준 결합도는 시스템에서 구현 의존성에 기인하는 결합도를 의미하는데, 예를 들면 서버 클래스의 수정은 클라이언트 클래스의 수정을 야기하는데 이 경우 클래스 수준 결합도를 갖는다.

Lee et al.[5]은 한 개체에서 다른 개체로 정보의 흐름이 있으면 그 두 개체 사이에 연결성이 존재한다고 주장하였다. 그래서 클래스 사이에 전달되는 정보의 양이 많을수록 결합도가 크다고 생각하고, 클래스 사이의 메소드 호출 회수와 인자 개수를 곱하여 더한 수를 정보 흐름 클래스 결합도(information flow-based class coupling)로 정의하였다. 또한 클래스 메소드의 각 내부 호출에 대하여 인자 개수에 호출 회수를 곱하고, 모든 내부 호출에 대해 그것을 합하여 클래스 응집도로 정의하였다. 정보 흐름 클래스 결합도와 마찬가지로 클래스 내부 메소드들 사이의 정보의 흐름이 많으면 클래스 응집도가 높다.

Allen et al.[7]은 개수 측정 방법이 아닌 정보 이론 방법(information-theory approach)을 이용하여 메트릭을 제안하였다. 정보 이론 방법에서는 기호(symbolic) 내용을 측정하여 패턴을 유도하며 일반적인 패턴일수록 적은 정보를 갖는다. 시스템 모듈 결합도는 모듈사이 에지 관계들의 최소 설명 길이(minimum description length)로 정의하는데, 결합도가 클수록 최소 설명 길이가 길다. 그리고 응집도는 모든 노드들이 서로 연결된 가장 응집력있는 시스템을 정의하고 이 시스템의 최소 설명 길이에 대한 현재 시스템의 최소 설명 길이 비율로 정의하였다.

Bieman and Kang[10]은 Chidamber and Kemerer 응집도와 유사한 TCC(Tight Class Cohesion)와 LCC(Loose Class Cohesion) 메트릭을 제안하였다. TCC는 직접적 또는 간접적으로 공통 속성을 사용하는 메소드 쌍

의 비율로 정의되고, LCC는 직접적 또는 간접적으로 연결된 메소드 쌍의 비율로 정의된다.

Chae and Kwon[11]은 접근자(accessor), 위임자(delegation), 생성자(constructor), 그리고 소멸자(destructor) 등의 특수 메소드(special method)를 배제한 상태에서 클래스의 가장 응집력 있는 형태 MCC(Most Cohesive form of Class)를 정하고, 그 클래스가 MCC에 유사한 정도를 응집도로 제안하였다. 이 응집도는 연결성 요소(connectivity factor)와 구조적 요소(structure factor)의 곱으로 계산하는데, 연결성 요소는 클래스 멤버들 사이의 연결 정도를 나타내고, 구조적 요소는 클래스 구성 요소들의 응집 정도를 나타낸다.

Briand et al.[12]는 개발 초기 단계 메트릭의 존재가 성공적인 소프트웨어 개발의 핵심 요소라고 주장하고, 상위 수준 설계(high-level design)을 위한 메트릭을 정의하고 검증하였다. 이 메트릭은 상위 수준 설계의 모듈 인터페이스에 나타난 자료 선언과 서브루틴 사이의 상호작용을 사용한 응집도와 결합도에 기반한다. Briand et al.[4]는 C++ 언어에서 영향의 방향(locus of impact), 관계(relationship), 그리고 상호작용(interaction)에 따라 클래스 결합도를 분류하였다. 영향의 방향에 따라 import/export로 구분하고, 관계를 상속, 친구(friendship), 기타 등의 관계로 분류하고, 상호작용을 클래스-속성, 클래스-메소드, 그리고 메소드-메소드의 상호작용으로 구분하였다.

Yacoub et al.[6]은 동적 복잡도와 실행 시나리오에 기반한 객체 결합도를 제안하였다. 병행 객체들로 구성된 응용 프로그램의 동적인 면을 연구하기 위하여 응용 프로그램 시나리오를 시뮬레이션할 수 있는 ROOM(Real-Time Object Oriented Modeling)을 사용하였다. Lee et al.[5]와 Yacoub et al.[6]의 결합도는 메시지가 한 클래스로 집중된 경우와 전체 메시지의 양은 같지만 여러 클래스로 분산된 경우를 구분하지 않는다.

그리고 Briand et al.[13,14]는 기존의 메트릭들을 종합하고 분석하여 알맞은 메트릭을 선택하거나 또는 가용한 메트릭이 없어 새로 정의할 필요가 있을 때 도움을 제공한다. 또한 Briand et al.[15]는 수학적 개념을 이용하여 정밀하고 특정 소프트웨어에 국한되지 않는 메트릭 프레임워크를 제안하였다. 이 프레임워크를 사용하여 크기, 길이, 복잡도, 결합도, 그리고 응집도 등의 중요한 메트릭 성질(property)을 정의하였다.

2.2 재구조화

Choi and Scacchi[16]는 모듈 사이에 자원 교환(resource exchange)을 나타내고, 시스템 재구조화 알고리즘을 적용하여 계층적인 설계를 유도하였다. 구현 명세로부터 설계 명세를 생성하기 위하여 구현 언어로 작성된 소스 코드를 분석하여 MIL(Module Interconnection Language)로 작성된 설계 명세를 생성한다. Resource-Flow Diagram(RFD)는 모듈 사이에 교환하는 자원을 사용하여 그들의 관계를 보여주고, Resource- Structure Diagram(RSD)는 모듈과 제어 모듈 사이의 제어 관계를 보여준다. 시스템 재구조화 알고리즘을 적용하여 모듈 결합도를 최소화하고, 변경 거리(alteration distance)를 최소화하면서 RFD로부터 RSD로 변환한다.

Abreu et al.[17]는 객체 지향 시스템 모듈화에 대한 정량적인 방법을 실험하여 검증하였다. 이 연구에서는 클래스들을 그룹화하기 위하여 통계적 방법인 클러스터 분석(cluster analysis)을 사용한다. 클러스터 분석을 위한 클래스 사이의 상이도(dissimilarity)는 6개의 등급 방식을 사용한 상대적 결합도를 사용하고, 각 결합도는 분류 영역에 따라 다른 가중치를 할당받는다. 이 결합도 자료는 MOODKit G2 도구를 사용하여 분석된다. 시스템 개발자가 제안한 포함 매트릭스와 클러스터 분석에 의해 만들어진 포함 메트릭스 사이에 매칭 알고리즘이 적용된다. 전체 클래스 쌍들의 성공 비율을 클러스터링에 대한 최종 매칭 값으로 계산한다. 최종 매칭 값이 높으면 시스템 개발자의 제안과 클러스터 분석 결과가 유사한 것을 의미하므로 재공학에 적용할 수 있다. 이 연구에서는 계층적이고 집적적인 클러스터링(hierarchical agglomerative clustering) 방법을 사용하는데, 알고리즘을 수행시키기 위하여 클러스터 개수를 시스템 개발자가 미리 정해야 한다.

Tesch and Klein[18]은 프로그램 또는 시스템을 계층적인 도표(hierarchy chart)로 나타내는 CAPO (Computer Assisted Process Organization) 방법을 사용하여 자료 흐름도와 시스템 사전으로부터 응집도와 결합도를 정의한다. 프로세스의 선행 관계를 나타내는 선행 행렬(precedence matrix), 프로세스가 동시에 호출될 수 있는지를 나타내는 시간 관계 행렬(matrix of timing relationship), 그리고 자료의 존재 유무를 나타내는 빈도 행렬(incidence matrix)을 정의하여 응집도를 계산하고, 결합도는 두 모듈에 사용된 자료 항목의 비율로 정의한다. 그리고 응집도와 결합도를 합성한 척도에 정수-프로그래밍(integer-programming)을 적용하여 설계를 최적화한다.

Kim et al.[19]은 자료 조각(data slices)과 유사한 처리 블록(processing block)을 정의하였는데, 이 처리 블록은 출력 변수와 자료 의존 관계 또는 제어 의존 관계를 가진 자료 토큰들의 집합이다. 그리고 응집도와 유사한 모듈 강도 개념을 이용하여 모듈을 분해하거나 합성한다. 즉,

모듈 강도가 낮으면 새로운 모듈들로 분할하고, 분할된 부품들이 패키지로 묶여진다. 부품들을 패키지로 묶기 위하여 모듈 강도 정보뿐만 아니라 모듈 함수와 설계 기준에 대한 이해가 필요하다.

Bieman and Kang[20]는 절차적(procedural) 소프트웨어 재구조화를 위한 정량적인 프레임워크, 즉 소프트웨어 설계 모델(models of software designs), 측정 기반의 재구조화 척도(measurement based restructuring criteria), 그리고 재구조화 과정(a process of restructuring)을 제공한다. 이 프레임워크에서 재구조화 과정은 설계 정보와 척도에 의하여 유도된다. 설계 정보는 시스템을 재구조화하기 위하여 코드로부터 추출되고, 설계 구조를 비교하기 위한 기준으로서 설계 응집도와 결합도를 사용한다. 재구조화는 시스템 요소들의 응집도를 올리고 결합도를 줄이도록 모듈의 분할과 합성을 통하여 수행된다.

3. 재구조화 과정을 위한 모델과 메트릭

3.1 전체적인 재구조화 과정

본 논문에서 제안하는 재구조화 과정(그림 1)의 추상 객체 지향 설계 정보는 객체 지향 모델링 도구 또는 객체 지향 언어로 작성된 코드에서 추출하여 초기 입력 자료로 사용된다. 추상 설계 정보로부터 메소드와 속성 사이 정보 교환 자료를 수집하여 메소드 사이 정보 교환 행렬을 만든다. 이 행렬을 사용하여 클래스 메트릭과 객체 지향 설계 메트릭을 유도하고, 이 메트릭을 사용한 적합도 함수를 기반으로 유전자 알고리즘을 적용한다. 유전자 알고리즘 수행 결과물는 초기 객체 지향 설계보다 품질이 개선된 설계이다.

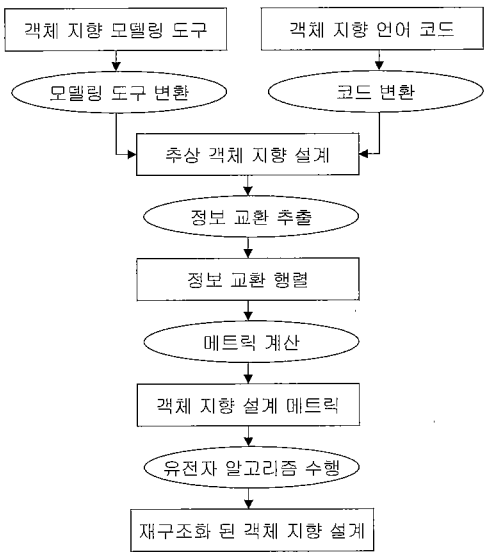


그림 1 객체 지향 설계 재구조화 과정

3.2 추상화 모델

객체 지향 시스템의 추상화 모델은 메소드, 속성, 클래스와 그들 사이의 관계를 간략히 나타내므로 메트릭 측정을 위하여 필수적이다. 본 논문에서는 객체 지향 설계를 CUG(Call-Use Graph)와 CAG(Class-Association Graph)로 나타낸다. CUG에서는 객체 지향 설계에 나타난 모든 클래스의 속성, 메소드, 그리고 그들 사이의 관계를 나타내고, CAG는 클래스들 사이의 관계를 요약하여 보여준다. 그래서 CUG와 CAG는 서로 보완적인 그래프 모델이다.

클래스의 메소드 사이에는 호출 관계(call relationship), 그리고 메소드와 속성 사이에는 사용 관계(use relationship)가 있다. 호출 관계는 메소드 사이에 메시지를 주고받는 관계이고, 사용 관계는 메소드가 속성 자료를 읽거나 쓰는 관계이다. 클래스 사이의 관계는 상속 관계와 상호작용 관계로 구분한다. 상속 관계는 부모 클래스의 속성과 동작을 상속받는 관계이고, 상호작용 관계는 상속 관계에 있지 않은 클래스 사이에 메시지를 보내거나 속성을 접근하는 관계이다.

[정의 1] 호출-사용 그래프(Call-Use Graph : CUG)

객체 지향 설계 D의 CUG는 방향성있는 그래프, $CUG_D = (V_U, E_U)$로 표현되며, V_U는 객체 지향 설계 D의 모든 속성과 메소드들의 집합이다. 그리고 $E_U = \{(x, y) \in V_U \times V_U | \ x \ calls \ or \ uses \ y\}$의 관계를 만족하는 가중치를 가진 에지(weighted edge)들의 집합이다. 메소드 사이의 호출 관계를 나타내는 에지의 집합을 E_c라 표시하면, 이 집합에 속하는 에지의 가중치는 정보 전달의 양, 즉 메소드 인자 개수+1로 표시한다 [5]. 1을 더하는 이유는 메시지를 받는 객체 자체를 계산에 포함하기 위해서이다. 메소드와 속성 사이의 사용 관계를 나타내는 에지의 집합을 E_u라 표시하면, 이 집합에 속하는 에지의 가중치는 1이다. 그리고 $E_U = E_c \cup E_u$이다. CUG에서 변수는 사각형으로 표시하고, 메소드는 원으로 표시한다. 호출 관계는 점선 화살표 그리고 사용 관계는 실선 화살표로 표시한다. 만약 $|args(M(x, y))|$가 x로부터 y로 가는 메시지 M의 인자 개수라고 할 때, CUG 에지는 다음의 가중치를 갖는다.

$$W(e_c(x, y)) = |args(M(x, y))| + 1, \ \text{if } e_c(x, y) \in E_c$$
$$= 0, \quad otherwise$$

$$W(e_u(x, y)) = 1, \ \text{if } e_u(x, y) \in E_u$$
$$= 0, \ otherwise$$

[정의 2] 클래스 연관 그래프(Class-Association

Graph : CAG)

객체 지향 설계 D의 CAG는 방향성있는 그래프, $CAG_D = (V_A, E_A)$로 표현되며, V_A는 객체 지향 설계 D의 클래스 집합이다. 그리고 E_A는 가중치를 가진 에지(weighted edge)들의 집합이고, 클래스 사이의 상속(inheritance) 관계를 나타내는 에지의 집합을 E_{ih}, 상호작용(interaction) 관계를 나타내는 에지의 집합을 E_{ia}라 표시하면 $E_A = E_{ih} \cup E_{ia}$이다. $E_{ih} = \{(x, y) \in V_A \times V_A |$ *Class x inherits from class y*$\}$이고 $E_{ia} = \{(x, y) \in V_A \times V_A |$ *a method of class x calls a method of class y, or uses an attribute of class y*$\}$이다. 에지 가중치는 CUG 가중치를 클래스 단계에서 합산한다. CAG에서 클래스는 사각형으로 표시하고 클래스 사이의 상속 관계는 끝이 둥근 실선으로 나타내고 상호작용 관계는 실선 화살표로 표시한다. 그리고 이 에지들에 가중치를 표시한다.

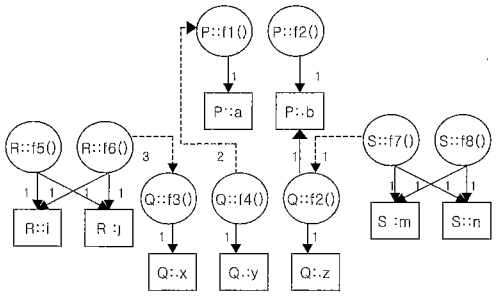
```
class P {
int          a, b;
public:
             int f1(int) { use a; }
             virtual int f2() { use b; }
};
class Q : public P {
int          x, y, z;
public:
             virtual int f2() { use b, z; }
             int f3(int, int) { use x; }
             int f4(int) { use y; call f1(s); }
};
class R {
int          i, j;
Q            q;
public:
             int f5(int) { use i, j; }
             int f6(int) { use i, j; call q.f3(s, t); }
};
class S {
int          m, n;
Q            q;
public:
             int f7(int) { use m, n; call q.f2(); }
             int f8(int) { use m, n; }
};
```

그림 2 객체 지향 설계 예

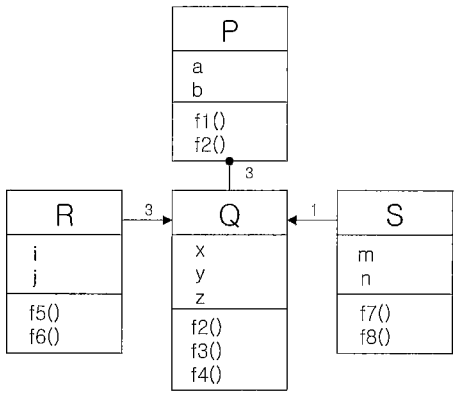
CUG와 CAG는 다음 성질을 만족하며, 앞으로 본 논문의 CUG와 CAG는 이 성질을 만족하는 CUG와 CAG를 말한다.

- CUG는 방향성 있는 비순환(acyclic) 그래프이고, CAG에서 상호작용 관계는 순환(cyclic)할 수 있지만 상속 관계는 비순환한다.
- CUG와 CAG의 각 노드는 유일한 이름을 가진다.

그림 2는 C++ 언어와 유사한 객체 지향 설계 예로서 "use"는 CUG 모델의 사용 관계, "call"은 호출 관계, 그리고 매개 변수 s와 t는 지역 변수를 나타낸다. 그림 2의 예를 CUG 모델과 CAG 모델로 표현하면 그림 3과 같다.



(a) CUG 모델



(b) CAG 모델

그림 3 그림 2의 추상화 모델

[정의 3] 메소드 정보 교환 행렬(Communication Matrix between Methods : CMM)

재구조화 과정을 위하여 CUG와 CAG 모델로부터 메소드 정보 교환 행렬(CMM)을 만든다. 이 행렬은 설계 D의 메소드 개수 차원 정방행렬이고, CUG 모델 에지 값을 사용하여 정의하므로 대칭 행렬이다. CMM의 각 요소는 메소드 사이의 정보 교환 양, 즉 CUG 모델의 메소드 사이 호출 관계 에지 가중치 $W(e_c(x,y))$와 두 메소드가 공통으로 접근하는 속성의 개수 $|share(x,y)|$를 더한 값을 나타낸다. 또한 CMM의 각 행은 한 메소드와 다른 모든 메소드들 사이의 정보 교환을 나타내므로 메소드의 동작 유형

(behavioral pattern)을 나타낸다. 클래스 생성자(constructor), 소멸자(destructor), 그리고 접근 메소드(access method)는 정보 교환 행렬에 포함시키지 않는다.

- $CMM(i,j) = CMM(j,i) = W(e_c(x,y)) + |share(x,y)|$, if $i = id(x)$ and $j = id(y)$
- $share(x,y) = \{z \mid x \text{ uses } z \text{ and } y \text{ uses } z\}$

여기서 $id(x)$는 메소드 x의 숫자로 된 식별자를 나타내고, 그림 3의 CUG 모델과 CAG 모델로부터 정보 교환 행렬을 구하면 표 1과 같다. 메소드 동작 유형인 CMM 행 벡터의 코사인(cosine) 척도[21]를 사용하여 메소드 사이의 유사성(SBM : Similarity Between Methods)을 정의하고, 메소드 동작 유사성으로부터 클래스 사이 유사성(SBC : Similarity Between Classes)과 클래스 사이 상속 유사성(ISBC : Inheritance Similarity Between Classes)을 정의한다.

표 1 정보 교환 행렬

메소드	P::f1()	P::f2()	Q::f2()	Q::f3()	Q::f4()	R::f5()	R::f6()	S::f7()	S::f8()
P::f1()	1(1:0)	0	0	0	2(0:2)	0	0	0	0
P::f2()	0	1(1:0)	1(1:0)	0	0	0	0	0	0
Q::f2()	0	1(1:0)	2(2:0)	0	0	0	0	1(0:1)	0
Q::f3()	0	0	0	1(1:0)	0	0	3(0:3)	0	0
Q::f4()	2(0:2)	0	0	0	1(1:0)	0	0	0	0
R::f5()	0	0	0	0	0	2(2:0)	2(2:0)	0	0
R::f6()	0	0	0	3(0:3)	0	2(2:0)	2(2:0)	0	0
S::f7()	0	0	1(0:1)	0	0	0	0	2(2:0)	2(2:0)
S::f8()	0	0	0	0	0	0	0	2(2:0)	2(2:0)

(괄호 안의 첫 번째 값은 공유 관계 값, 두 번째 값은 호출 관계 값, 괄호 바깥 값은 합)

[정의 4] 메소드 유사성, 클래스 사이 유사성, 그리고 클래스 사이 상속 유사성

$$SBM(m_1, m_2) = \frac{\sum_{k=1}^{M} CMM(i,k) * CMM(j,k)}{\sqrt{\sum_{k=1}^{M} CMM(i,k)^2} * \sqrt{\sum_{k=1}^{M} CMM(j,k)^2}}, \text{ if}$$

$i = id(m_1)$ and $j = id(m_2)$

$$SBC(C_i, C_j) = \sum_{p=1}^{|C_i|} \sum_{q=1}^{|C_j|} SBM(m_{ip}, m_{jq})$$

$$ISBC(C_i, C_j) = \sum_{p=1}^{|C_i|} \sum_{q=1}^{|C_j|} \begin{cases} SBM(m_{ip}, m_{jq}), & \text{if } CMM(id(m_{ip}), id(m_{jq})) \neq 0 \\ 0, & otherwise \end{cases}$$

이 식에서 M은 객체 지향 설계 D에 있는 메소드 개수, $SBM(m_{ip}, m_{jq})$은 클래스 C_i의 $p\text{-}th$ 메소드와 클래스 C_j의 $q\text{-}th$ 메소드 사이의 유사성, $|C_i|$는 클래스 C_i의 메소드 개수를 나타낸다.

3.3 메트릭(Metrics)

본 연구에서는 객체 지향 설계를 측정하기 위하여 정보 교환 행렬에 기반한 클래스 유사성을 사용하여 응집도와 결합도 메트릭을 정의한다.

3.3.1 응집도(Cohesion)

클래스 응집도는 클래스내 구성 요소들의 긴밀 정도를 의미하므로, 본 논문에서는 $SBC(C_i, C_i)$를 사용하여 클래스 메소드 동작 유사성으로 클래스 응집도를 계산한다. 객체 지향 설계 응집도는 클래스 응집도의 합으로 정의한다.

[정의 5] 클래스 응집도($COH(C)$)와 객체 지향 설계 응집도($COH(D)$)

$$COH(C_i) = \frac{SBC(C_i, C_i)}{|C_i| * |C_i|}, \quad i = 1, \ldots, n \quad \text{n : 객체 지향 설계 D에 존재하는 클래스 개수}$$

$$COH(D) = \sum_{C_i \in D} COH(C_i)$$

표 2 응집도 성질

1. 비음수성과 정규화(Nonnegativity and Normalization)
   - $COH(D)$는 최소 0, 최대 Max 값을 갖는다.
2. 널 값(Null value)
   - 객체 지향 설계의 모든 메소드 사이에 정보 교환이 없고 어느 두 메소드도 공통으로 다른 메소드와 정보 교환을 하지 않으면 $COH(D)$는 0 값을 갖는다.
3. 단조성(Monotonicity)
   - 클래스 멤버 사이에 정보 교환이 추가되면 $COH(D)$이 감소하지 않는다.
4. 응집력있는 클래스(Cohesive Classes)
   - 정보 교환이 없는 두 클래스 C_i와 C_j를 합병하여 새로운 클래스 $C_{i\cup j}$를 만들면 새로운 객체 지향 설계 D′의 $COH(D')$는 이전 설계 D의 $COH(D)$보다 크지 않다.

본 논문에서 제안한 응집도의 타당성을 검증하기 위하여 Briand[15]가 제안한 성질(표 2)을 이용하여 $COH(D)$를 평가한다. Briand 성질의 모듈과 모듈화 시스템은 본 연구의 클래스와 객체 지향 설계로 각각 가정한다. 응집도 정의가 메소드 벡터 각의 코사인 유사성으로부터 출발하므로 $COH(D)$는 비음수성을 갖고 0과 n사이 값으로 정규화되므로 첫 번째 성질 비음수성과 정규화를 만족한다. 클래스에 속한 모든 메소드들 사이에 정보 교환이 없고 어느 두 메소드도 공통으로 다른 메소드와 정보 교환을 하지 않으면 $COH(C)$가 0이다. 그리고 모든 클래스가 이와 같은 성질을 가지면 $COH(D)$가 0이다. 따라서 두 번째 성질 0의 값을 만족한다. 클래스 메소드 사이에 정보 교환이 추가되면 $SBM(m_1, m_2)$이 증가하여 $COH(C)$이 증가되므

로 $COH(D)$가 증가한다. 따라서 세 번째 성질 단조성을 만족한다. 직접적인 정보 교환이 없는 클래스라도 다른 메소드와 공통으로 정보 교환을 하면 간접적으로 정보 교환을 한다. 예를 들면 그림 3의 (a)에서 클래스 P와 S는 직접적으로 정보 교환이 없으나 클래스 P의 f2()와 클래스 S의 f7(), 두 메소드가 클래스 Q의 f2()와 공통으로 정보 교환을 하므로 클래스 P와 S는 간접적으로 정보 교환을 한다. 이러한 두 클래스 C_i와 C_j를 합병하면 간접 정보 교환으로 인해 합병된 클래스의 $SBC(C_iUj, C_iUj)$가 증가할지라도 새로 합병된 클래스의 크기가 훨씬 커지므로 $COH(C_iUj)$는 $COH(C_i)$와 $COH(C_j)$의 합보다 작다. 따라서 새로운 객체 지향 설계의 $COH(D')$는 합병전의 $COH(D)$보다 크지 않다. 따라서 네 번째 성질 응집력있는 클래스도 만족한다.

3.3.2 결합도(Coupling)

결합도는 클래스와 다른 클래스 사이의 연결 강도를 의미하므로, 본 연구에서는 연결 강도를 클래스 사이 유사도를 사용하여 측정한다. 클래스 사이 유사도는 두 클래스 사이의 모든 메소드 쌍의 유사도 합으로 정의하므로 두 클래스에 속한 메소드들의 동작 유형이 유사할 수록 두 클래스의 결합도가 크다고 간주한다. 본 논문의 결합도는 상속 결합도와 상호작용 결합도로 구성된다. 객체 지향 설계에서 상속 관계는 재사용을 실현하는 설계 방법으로서 개발 생산성과 신뢰성을 제공한다. 클래스 상속 결합도는 상속 계층상의 클래스들 사이에 연결 강도를 의미하므로 일반적인 설계 휴리스틱, 즉 응집도는 높을수록 좋고 결합도는 낮을수록 바람직하다[23]는 통념과 달리 적당히 높은 것이 바람직하다. 반면에 상호작용 결합도는 상속 관계가 아닌 클래스 사이의 연결 강도를 의미하므로 낮을수록 바람직하다. 그리고 객체 지향 설계 D의 상속 결합도와 상호작용 결합도는 클래스 상속 결합도의 합과 상호작용 결합도의 합으로 각각 정의한다.

[정의 6] 클래스 상속 결합도($COP_{IH}(C)$)와 클래스 상호작용 결합도($COP_{IA}(C)$)

$$COP_{IH}(C_i) = \sum_{C_j \in Ancestors(C_i)} ISBC(C_i, C_j)$$

$$COP_{IA}(C_i) = \sum_{C_j \in Interacts(C_i)} SBC(C_i, C_j)$$

$Ancestors(C_i)$: 상속 계층에서 클래스 C_i의 상위 계층에 있는 클래스 집합

$Interacts(C_i)$: 클래스 C_i의 상속 계층상에 있지 않고 클래스 C_i와 메시지 교환 또는 속성 접근을 교환하는 클래스 집합

[정의 7] 객체 지향 설계 상속 결합도($COP_{IH}(D)$)와 상호작용 결합도($COP_{IA}(D)$)

표 3 결합도 성질

1. 비음수성(Nonnegativity)
   · $COP_{IA}(D)$와 $COP_{IH}(D)$는 음수가 아니다.
2. 널 값((Null value)
   · 클래스 사이에 정보 교환이 없으면 $COP_{IA}(D)$와 $COP_{IH}(D)$는 0이다.
3. 단조성(Monotonicity)
   · 클래스 사이에 정보 교환이 추가되면 $COP_{IA}(D)$와 $COP_{IH}(D)$이 감소하지 않는다.
4. 클래스 합병(Merging of classes)
   · 두 클래스 C_i와 C_j를 합병하여 새로운 클래스 C_iUj를 만들면 새로운 객체 지향 설계 D'의 $COP_{IA}(D')$와 $COP_{IH}(D')$는 각각 이전 설계 D의 $COP_{IA}(D)$와 $COP_{IH}(D)$보다 크지 않다.
5. 서로 소인 클래스 합병(Disjoint class additivity)
   · 정보 교환이 없는 클래스 C_i와 C_j를 합병하여 클래스 C_iUj를 만들면 새로운 객체 지향 설계 D'의 $COP_{IA}(D')$와 $COP_{IH}(D')$는 각각 이전 설계 D의 $COP_{IA}(D)$와 $COP_{IH}(D)$와 같다.

$$COP_{IH}(D) = \sum_{C_i \in D} COP_{IH}(C_i)$$

$$COP_{IA}(D) = \sum_{C_i \in D} COP_{IA}(C_i)$$

본 논문에서 제안한 결합도의 타당성을 검증하기 위하여 Briand[15]가 제안한 결합도 성질(표 3)을 이용하여 평가한다. $COP_{IH}(D)$와 $COP_{IA}(D)$는 메소드 벡터 코사인 값 $SBM(m_1,m_2)$에 기반하므로 음수 값을 갖지 않는다. 따라서 첫 번째 성질 비음수성을 만족한다. 클래스 C_i가 조상 클래스가 아닌 다른 클래스와 정보 교환이 없으면 $Interacts(C_i)$이 공집합이므로 $COP_{IA}(C_i)$는 0의 값을 갖는다. 그리고 클래스 C_i가 조상 클래스와 정보 교환이 없으면 $COP_{IH}(C_i)$이 0이다. 그리고 모든 클래스의 COP_{IH}와 COP_{IA}가 0이면 $COP_{IH}(D)$와 $COP_{IA}(D)$도 0이다. 따라서 두 번째 성질 0의 값을 만족한다. 상호작용 관계에 있거나 또는 상속 관계에 있는 두 클래스 C_i와 C_j 사이에 정보 교환이 추가되면 $SBC(C_i,C_j)$ 또는 $ISBC(C_i,C_j)$가 증가하므로 클래스의 COP_{IA} 또는 COP_{IH}의 값이 각각 증가한다. 따라서 $COP_{IA}(D)$ 또는 $COP_{IH}(D)$의 값도 각각 증가하므로 세 번째 성질 단조성을 만족한다. 두 클래스 C_i와 C_j를 합병하면 두 클래스 사이 정보 교환이 합병된 클래스 C_iUj의 내부 정보 교환으로 전환된다. 따라서 새로 합병된 클래스의 $COP_{IH}(C_iUj)$와 $COP_{IA}(C_iUj)$는 각각 합병전의 $COP_{IH}(C_i)$와 $COP_{IH}(C_j)$의 합, $COP_{IA}(C_i)$와 $COP_{IA}(C_j)$의 합보다 작거나 같다. 그러므로 새로운 클래스를 포함한 객체 지향 설계의 $COP_{IH}(D')$와 $COP_{IA}(D')$은 각각 합병전의 $COP_{IH}(D)$와 $COP_{IA}(D)$보다 크지 않

으므로 네 번째 성질 클래스 합병도 만족한다. 응집도 네 번째 성질 설명에서와 같이 직접적인 정보 교환이 없더라도 간접적으로 정보 교환이 가능하다, 이러한 두 클래스 C_i와 C_j를 합병하면 간접적인 정보 교환이 새로운 클래스 C_{iUj}의 내부 정보 교환으로 전환된다. 따라서 새로운 클래스의 $COP_{IH}(C_{iUj})$와 $COP_{IA}(C_{iUj})$는 각각 합병전의 $COP_{IH}(C_i)$와 $COP_{IH}(C_j)$의 합, $COP_{IA}(C_i)$와 $COP_{IA}(C_j)$의 합보다 작고 또한 새로운 객체 지향 설계의 $COP_{IH}(D')$와 $COP_{IA}(D')$도 각각 합병전의 $COP_{IH}(D)$와 $COP_{IA}(D)$보다 작다. 그러므로 다섯 번째 성질 서로 소인 클래스의 합병은 만족하지 못한다.

4. 객체 지향 설계 재구조화

본 장에서는 설계 모델 검증을 위한 재구조화의 의미와 유전자 알고리즘을 적용한 재구조화에 대해서 기술한다.

4.1 재구조화의 의미(semantics)

객체 지향 스키마 변형의 의미[24]는 객체 지향 소프트웨어의 동작 측면보다 패러다임에 기반한 상속 자료 모델에 초점을 맞춘다. 본 연구는 재구조화 과정에서 구성 요소인 메소드, 속성, 그리고 동작을 나타내는 그들 사이의 관계 에지가 새로 추가되거나 삭제되지 않고 그대로 보존되는 것이 목적이고 다음과 같이 정의한다.

[정의 8] 객체 지향 설계 보존

객체 지향 설계 D가 $CUG_D = (V_U, E_U)$, $CAG_D = (V_A, E_A)$로 표시될 때 CAG에서 하위 클래스가 없는 클래스 집합을 $Leaf(CAG_D) = \{v \mid Any\ x \in V_A\ does\ not\ inherit\ from\ v\}$라 하자. $Leaf(CAG_D)$ 클래스들과 그들 사이의 에지로 구성한 CAG를 $LCAG_D = (LV_A, LE_A)$라 하고, $Leaf(CAG_D)$ 클래스가 상속받은 멤버를 포함하여 가지고 있는 모든 메소드, 속성, 그리고 그들 사이의 에지로 구성한 CUG를 $LCUG_D = (LV_U, LE_U)$라 하자. 그리고 새로운 객체 지향 설계 D'의 $LCUG_{D'} = (LV_U', LE_U')$, $LCAG_{D'} = (LV_A', LE_A')$라 하자. 만약 D와 D'가 아래 두 성질을 만족하면 D'는 객체 지향 설계 D를 보존한다고 정의한다.

- $n(LV_U) = n(LV_U')$, $n(LE_U) = n(LE_U')$
- For $\forall\ e(x,y) \in LE_U$, $\exists\ e'(w,z) \in LE_U'$ such that $W(e(x,y)) = W(e'(w,z))$

[정의 9] 객체 지향 설계 재구조화

CUG와 CAG로 표시한 객체 지향 설계 D의 변경을 R이라 할 때 $\Phi_R = \{(D, D') \mid D'\ can\ be\ obtained\ from\ D\ by\ R\}$이라 한다. 모든 (D, D') $\in \Phi_R$ 에 대하여 D'가 설계 D를 보존한다면 R을 객체 지향 설계 재구조화라고 정의한다.

4.1.1 기본 재구조화 행위

설계 모델 CUG와 CAG에 대한 8 개의 기본 재구조화 행위를 정의한다. 정의한 행위는 CUG와 CAG 노드 이름 변경, 클래스 에지 가중치 변경, 클래스 에지 추가/삭제, 메소드 또는 속성의 이동, 그리고 클래스 노드 추가/삭제 등이다.

행위 1 CUG 노드 이름 변경 : CUG에서 다른 노드와 이름이 중복되지 않도록 이름을 변경한다.

행위 2 CAG 노드 이름 변경 : CAG에서 다른 노드와 이름이 중복되지 않도록 이름을 변경한다.

행위 3 클래스 에지 가중치 변경 : 클래스에 속한 메소드 또는 속성들 사이의 호출 에지와 사용 에지를 더하여 CAG의 클래스 사이 상호작용 에지와 상속 에지 가중치를 갱신한다.

행위 4 클래스 에지 추가 : 상속 에지가 추가되면 자식 클래스는 부모 클래스의 메소드와 속성을 상속받는다. 만약 상속 에지가 순환하면 그 추가 요구는 취소된다. 상속 관계에 있지 않은 두 클래스의 멤버 사이에 사용 에지 또는 호출 에지가 존재해야 그 클래스들 사이에 상호작용 에지가 추가될 수 있다. 추가된 에지 가중치는 행위 3에서와 같이 계산된다.

행위 5 클래스 에지 삭제 : 상속 에지가 삭제되면 자식 클래스는 더 이상 부모 클래스의 멤버들을 상속받지 않는다. 만약 상속 관계에 있지 않은 두 클래스 멤버 사이에 호출 또는 사용 에지가 없으면 두 클래스 사이의 상호작용 에지가 삭제된다.

행위 6 메소드 또는 속성의 이동 : 에지를 가진 멤버의 이동으로 인해 멤버 이름이 중복되거나 또는 클래스 사이 에지가 추가 또는 삭제되기도 한다. 이름 중복은 행위 1에 의하여 해결되고, 에지 추가는 행위 4 그리고 에지 삭제는 행위 5에 의하여 수행된다. 또한 이동은 클래스 사이 에지 가중치 갱신을 요구하나(행위 3), 멤버 사이 에지 가중치는 변경하지 않는다.

행위 7 클래스 노드 추가 : 멤버가 없는 클래스 노드를 이름이 중복되지 않도록 추가한다.

행위 8 클래스 노드 삭제 : 메소드와 속성이 하나도 없으면 클래스 노드를 삭제한다. 클래스 노드를 삭제하면 행위 5를 적용하여 연결된 에지도 함께 삭제한다.

4.1.2 재구조화 행위의 견고성(soundness), 완전성(completeness), 그리고 최소성(minimality)

견고성은 모든 재구조화 행위 결과가 객체 지향 설계를 보존하는 것을 의미하고 완전성은 기본 재구조화 행위 집합이 모든 가능한 객체 지향 설계 재구조화를 수행하는 것을 의미한다. 그리고 최소성은 기본 재구조화 행위의 최소

부분 집합은 다른 행위들을 사용하여 유도될 수 없는 것을 의미한다.

[정리 1] (견고성) 모든 기본 재구조화 행위는 객체 지향 설계를 보존한다.

증명. 기본 재구조화 행위를 CUG와 CAG에 적용했을 때, 결과 그래프가 설계를 보존함을 보인다.

• 행위 1 CUG 노드 이름 변경

CUG 노드 이름 변경은 LCUG 노드 개수, 에지 개수, 그리고 에지 가중치에 영향을 미치지 않는다.

• 행위 2 CAG 노드 이름 변경

행위 1과 유사.

• 행위 3 클래스 사이 에지 가중치 변경

CAG에서 클래스 사이 에지 가중치 변경은 LCUG 노드 개수와 에지 개수에 영향을 미치지 않고 LCUG 에지 가중치도 변경하지 않는다.

• 행위 4 클래스 사이 에지 추가

CAG에서 클래스 사이 상호작용 에지 추가는 LCUG 노드 개수와 에지 개수에 영향을 미치지 않고 LCUG 에지 가중치도 변경하지 않는다. CAG에서 상속 관계 에지 추가는 Leaf(CAG)를 변경시키므로 LCAG의 노드 집합에 영향을 미치나 LCUG 노드 집합에는 영향을 미치지 않는다. 왜냐하면 상속 관계 추가가 Leaf(CAG)에 속한 클래스의 전체 메소드와 속성 집합에는 영향을 미치지 않기 때문이다.

• 행위 5 클래스 에지 삭제

행위 4와 유사

• 행위 6 메소드 또는 속성의 이동

다른 클래스로의 이동은 LCUG 노드 개수, 에지 개수 그리고 에지 가중치에 영향을 미치지 않는다. 그리고 이동시킨 후에 클래스 에지 가중치 갱신(행위 3)은 설계를 보존하고 이름을 변경해야 하는 경우(행위 1)도 설계를 보존한다.

• 행위 7 클래스 노드 추가

CAG에서 클래스 노드 추가는 LCUG 노드 개수, 에지 개수, 그리고 에지 가중치에 영향을 미치지 않고 이름이 중복된 경우에 해결하는 방법도 행위 2에 의하여 설계 보존한다.

• 행위 8 클래스 노드 삭제

멤버 메소드와 속성이 없으면 클래스를 삭제하므로 LCUG 노드 개수, 에지 개수, 그리고 에지 가중치에 영향을 미치지 않는다. 그리고 클래스 노드 삭제 후의 클래스 에지 삭제(행위 5)도 설계를 보존한다. ■

[정리 2] (완전성) CUG와 CAG로 표시한 임의의 설계 D와 D′가 객체 지향 설계를 보존할 때, D로부터 D′를 생성하는 일련의 기본 재구조화 행위가 존재한다.

증명. 다음과 같이 기본 재구조화 행위만을 사용하여 D를 D′로 재구조화한다.

1. D′에 존재하는 클래스를 D에 추가하기 위하여 행위 7을 사용하고 이름 변경은 행위 2를 사용한다. 이 행위는 D에 존재하지 않는 D′의 클래스가 모두 추가될 때까지 반복한다.

2. D′의 메소드와 속성에 대응하는 D의 메소드와 속성에 대하여 행위 6을 사용하여 D의 해당 클래스로 이동시킨다. 이름을 변경해야 하는 경우에는 행위 1을 사용한다. 이 행위는 모든 메소드와 속성에 대하여 반복하여 수행한다.

3. D′에 존재하지 않는 D의 클래스들을 삭제하기 위하여 행위 8을 사용한다. 관련된 에지를 삭제하기 위하여 행위 5를 사용한다. 이 행위는 D′에 존재하지 않는 클래스가 D로부터 모두 삭제될 때까지 수행한다.

4. D에서 클래스들의 메소드 또는 속성 사이에 에지가 있으면 행위 4를 사용하여 클래스 사이에 에지를 추가한다.

5. D의 모든 클래스 사이 에지에 행위 3을 적용하여 에지 가중치를 갱신한다.

위와 같은 순서로 재구조화 행위를 적용하면 D′는 객체 지향 설계 D를 보존한다. 왜냐하면 D는 D′와 같은 멤버를 가진 클래스와 같은 가중치를 가진 에지를 모두 포함하고 있고 더 이상의 클래스와 에지는 없기 때문이다. ■

[정리 3] (최소성) 기본 재구조화 행위의 최소 부분 집합은 {행위 1, 행위 4, 행위 5, 행위 6, 행위 7, 행위 8}이다.

증명.

- 행위 1이외에 어떤 재구조화 행위 집합도 CUG 노드 이름을 변경할 수 없다.
- 행위 2는 행위 6, 행위 7, 그리고 행위 8의 조합으로 유도될 수 있다.
- 행위 3은 행위 4와 행위 5의 조합으로 유도될 수 있다.
- 행위 4이외에 어떤 재구조화 행위 집합도 클래스 사이 에지를 추가할 수 없다.
- 행위 5이외에 어떤 재구조화 행위 집합도 클래스 사이 에지를 삭제할 수 없다.
- 행위 6이외에 어떤 재구조화 행위 집합도 클래스 멤버를 다른 클래스로 이동시킬 수 없다.
- 행위 7이외에 어떤 재구조화 행위 집합도 클래스 노드를 추가할 수 없다.
- 행위 8이외에 어떤 재구조화 행위 집합도 클래스 노드를 삭제할 수 없다. ■

4.2 유전자 알고리즘을 적용

유전자 알고리즘을 이용한 재구조화 과정은 메트릭을 사용하여 객체 지향 설계 구성 요소들을 클러스터링하는

과정이다. 클래스 응집도가 좋지 않은 클래스의 구조를 변경함으로써 좋은 메트릭 값을 가진 클래스들로 재구성하는 것이다. 이렇게 구조를 변경할 때 클래스 자체뿐만 아니라 전체 객체 지향 설계도 같이 고려한다. 유전자 알고리즘을 이용한 객체 지향 설계 재구조화 과정은 다음과 같다.

1. 초기 객체 지향 설계 D의 응집도와 결합도 메트릭을 계산한다.
2. 객체 지향 설계 D의 클래스 중에서 응집도가 가장 낮은 클래스를 선택한다.
3. 전체 설계 응집도나 결합도가 향상되도록 단계 2에서 찾은 클래스 멤버를 다른 클래스 또는 새로운 클래스로 이동시켜 분해한다.
4. 전체 설계 응집도나 결합도가 향상되도록 단계 3의 분해된 클래스와 다른 클래스를 결합한다.
5. 전체 설계 응집도나 결합도가 향상되지 않을 때까지 단계 2, 3, 4를 반복한다.

단계 3과 4에서 클래스를 분해하고 합성할 때에 기본 재구조화 행위들이 사용되며 결합도와 응집도를 최적화하기 위하여 유전자 알고리즘을 적용한다. 즉, 단계 3과 4에서 응집도가 낮은 클래스를 어떻게 분해하고 어느 클래스와 합성하는 것이 전체 객체 지향 설계 응집도와 결합도를 최적화하는 방법인지 찾기 위하여 유전자 알고리즘을 적용하는 것이다. 이러한 재구조화는 개발자의 설계 작업을 보조하는 역할을 하므로 재구조화 과정 중에 개발자의 판단이 필수적이다. 예를 들어 전체 설계 응집도나 결합도 값은 향상되지만 프로그램 또는 클래스 이해성 등의 다른 중요한 요소를 고려할 때 특정 클래스를 분해 또는 합성하는 것이 바람직하지 않을 경우가 있기 때문이다.

```
(1) compute similarity for each method pair;
(2) compute cohesion and coupling of an initial design;
(3) do {
(4)   select a class with the lowest cohesion;
(5)   create an initial population of fixed size p;
(6)   compute the fitness of the population;
(7)   do {
(8)         select parent1 and parent2 from population;
(9)     offspring = crossover(parent1, parent2);
(10)    mutation(offspring);
(11)    local-improvement(offspring);
(12)        compute fitness of offspring
(13)        replace(population, offspring);
(14)    } while (population is not converged and
                  generations < gmax);
(15) } while ( there exists a class to be restructured );
(16) return the best solution;
```

그림 4 유전자 알고리즘을 적용한 재구조화 방법

유전자 알고리즘[22]은 선택적 도태나 돌연변이와 같은 생물 진화의 원리로부터 착안된 비결정론적(non-deterministic) 알고리즘으로서 시간에 따라 진화하는 시스템을 모델링하거나, 조합 최적화 문제(combinatorial optimization problem)에 주로 사용된다. 유전자 알고리즘을 적용하기 전에 먼저 문제에 맞는 염색체(chromosome)의 구조를 정의하고 그 염색체의 적합도를 계산하는 함수를 설계한 후에 유전자 연산을 정의한다. 유전자 알고리즘을 실행시키기 위하여 염색체들의 모임인 모집단(population)을 생성한 후에 적합도 함수를 사용하여 염색체들을 평가한다. 그리고 적합도에 비례하는 확률을 사용하여 염색체들을 선택(selection)하여 교배(crossover)시키고, 또 변이(mutation)시킨 후에 모집단의 품질이 낮은 염색체를 대체(replacement)하는데, 이 과정이 그림 4에 나타나 있다. 이런 과정이 세대가 흘러가면서 반복되어 적합도가 높은 염색체들이 모집단의 많은 부분을 차지하면 이 중에서 가장 좋은 적합도 함수를 가진 염색체를 해(solution)로 선택한다. 그러나 유전자 알고리즘은 실행 시간이 길어질 수 있으며, 또 지역 최적해 근처에서 지역 최적해를 찾는데 효율적이지 못한 특징이 있어 일부 유전자 알고리즘에서는 이 부분을 보완해주는 지역 미세 조정 방법(local-improvement)을 사용한다. 본 논문의 재구조화 과정이란 객체 지향 설계 응집도와 결합도에 따라 메소드와 속성들을 최적화된 클래스들 $D=\{C_1, C_2, C_3, C_4, \ldots, C_k\}$로 나누는 것을 말하며, 이 과정에서 $COP_{IA}(D)$는 최소화되고, $COH(D)$와 $COP_{IH}(D)$는 최대화되도록 적합도 함수를 다음과 같이 정의하여 이 함수 값을 최소화하는 클래스들을 찾는 것이다.

[정의 10] 유전자 알고리즘을 위한 적합도 함수 (Fitness(D))

$$Fitness(D) = \frac{1}{COH(D)} + \frac{COP_{IA}(D)}{COP_{IH}(D)}$$

유전자 알고리즘에서 각각의 해는 알고리즘이 이해할 수 있는 염색체로 표시된다. 하나의 염색체는 클래스 메소드 개수만큼의 유전자(gene)들로 구성되며, 각 유전자는 메소드에 해당되고 유전자의 값은 메소드가 속한 클래스를 나타낸다. 그래서 염색체는 N-정수 스트링으로 구성되며, i 번째 유전자는 선택된 클래스의 i 번째 메소드가 속한 클래스 식별자를 나타낸다. 예를 들면 그림 5에서 첫 번째 메소드는 클래스 1에 속하고, 세 번째 메소드는 클래스 2에 속한 것을 나타낸다. 그래서 그림 5는 클래스 1에 메소드 1, 7, 9, 클래스 2에 메소드 2, 3, 10, 13이 속한 것을 나타낸다. 유전자 알고리즘은 응집도가 낮은 클래스의 메소드들을 다른 클래스들로 랜덤하게 할당한 염색체

메소드	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
클래스	1	2	2	4	5	4	1	3	1	2	4	3	2	3	4

그림 5 재구조화를 위한 염색체 표현

들의 모임인 초기 모집단을 생성한 후에 진화시킨다. 그리고 클래스 속성은 그 속성과 정보 교환이 가장 많은 클래스로 이동시킨다.

변이 유전 연산자를 적용한 후에 적합도 함수를 이용한 미세 조정 휴리스틱(그림 6)을 적용한다. 이 휴리스틱은 일종의 언덕-오르기 방법(hill-climbing)으로 가상 함수가 아닌 메소드 중에서 선택된 메소드와 가장 유사한 메소드를 찾는다. 만약 두 메소드가 같은 클래스의 메소드가 아니면 두 메소드를 다른 클래스로 이동시키고 적합도 함수 값이 개선되면 우수한 해를 찾은 것이므로 받아들이고 그렇지 않으면 원래 클래스로 이동시킨다. 본 논문에서는 우수한 품질의 해를 찾고 수행 시간도 너무 많이 소비하지 않기 위하여 간단한 형태의 지역 미세 조정을 사용하였다.

```
for( each method in the selected class )
    find the most similar method, not a virtual method,
among all methods;

let m1 and m2 be two methods of the most similar pair
among the above pairs;
if ( class(m1) ≠ class(m2) ) {
    move m1 and m2 to another class;
    if( fitness(D) is not improved )
        reset m1 and m2;
}
```

그림 6 지역 미세 조정 휴리스틱

본 유전자 알고리즘을 이용한 재구조화 방법의 시간 복잡도(time complexity)를 분석하면 다음과 같다. 먼저 객체 지향 설계 D의 전체 메소드 개수를 m, 전체 클래스 개수를 n, 모집단의 크기를 p, 최대 세대 반복수를 g_{max}라고 가정한다. 전체 클래스 개수 n은 재구조화 중간에 변할 수 있으나 모두 n으로 나타낸다. 그림 4의 단계 (1)은 $O(m^2)$, 단계 (2)는 $O(m+n)$, 단계 (4)의 복잡도는 $O(n)$, 단계 (5)는 $O(p*m/n)$, 단계 (6)은 다음과 같은 과정으로 수행된다.

(a) 각 클래스 메소드로부터 전체 설계의 메소드 배열을 만든다 : $O(m)$

(b) 이 배열로부터 클래스들의 상호작용 관계와 상속 관계를 설정한다 : $O(m^2)$

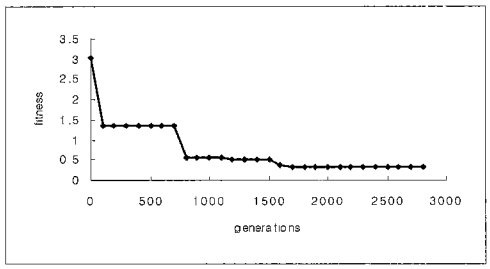
(c) 각 클래스에 대하여 COH, COP_{IH}, 그리고 COP_{IA}를 계산한다. : $O(m^2)$

따라서 단계 (6)의 모집단 적합도 계산의 시간 복잡도는 $O(p*m^2)$이다.

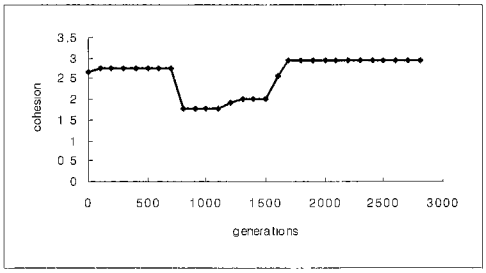
유전자 알고리즘 연산자중에서 단계 (8)의 선택 연산은 $O(p)$, 단계 (9)의 교배 연산은 $O(m/n)$, 단계 (10)의 변이 연산은 $O(m/n)$, 단계 (11)의 지역 미세 조정은 $O(m^2/n)$, 단계 (12)의 적합도 계산은 $O(m^2)$, 단계 (13)의 대체 연산은 $O(p)$이다. 따라서 유전자 알고리즘 한 세대의 시간 복잡도는 $O(p + m^2)$로 단순화된다. 그러므로 본 알고리즘의 시간 복잡도는 $O(m^2+n*(p*m^2 + g_{max}(p + m^2))$이나 모집단의 크기 p는 상수이고, m에 비하여 알고리즘 수행시간에 미치는 영향이 작으므로, 시간 복잡도는 $O(g_{max}*n*m^2)$로 단순화된다. 그러나 객체 지향 설계의 각 메소드가 속할 수 있는 모든 클래스를 방문하는 exhausive search 알고리즘의 시간 복잡도는 $O(m^n)$이다.

4.3 재구조화 수행 결과

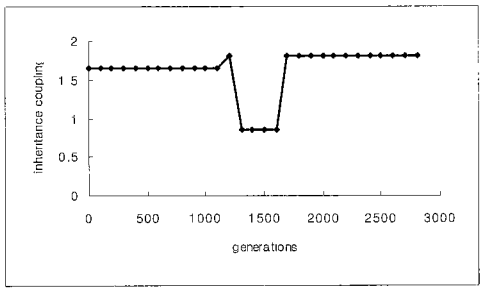
그림 7은 표 1의 정보 교환 행렬 정보를 사용한 유전자 알고리즘 수행 과정에서 우수 해의 각 그래프이다. 그림 7의 (a)는 세대가 지나가면서 적합도 함수가 줄어들어 수렴하는 것을 보여주는데, 이것은 세대가 반복됨에 따라 유전자 알고리즘이 우수한 해를 찾는 것을 의미한다. 응집도(그림 7의 (b))는 세대가 지나가면서 중간에 감소하다가 다시 증가하여 최종적으로 초기 값보다 큰 값을 가진다.



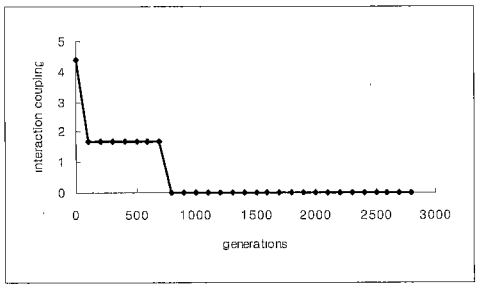
(a) Fitness(D)



(b) COH(D)



(c) $COP_{IH}(D)$



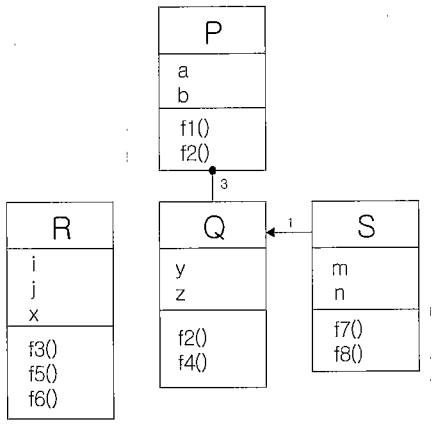
(d) $COP_{IA}(D)$

그림 7 유전자 알고리즘 수행 중에서 우수 해의 각 그래프

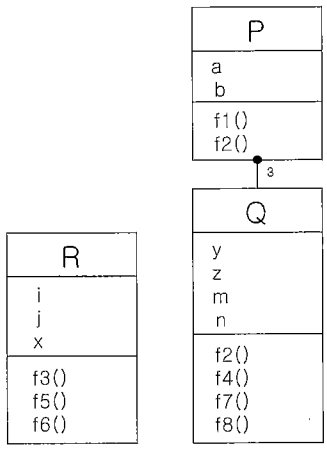
상속 결합도(그림 7의 (c))도 응집도와 마찬가지로 중간에 감소하다가 증가하였다. 상호작용 결합도(그림 7의 (d))는 세대가 지나가면서 계속 감소하여 수행 중간에 0이 되었다.

그림 7의 과정을 CUG와 CAG 모델로 표시하면 그림 8과 그림 9이다. 그림 8은 유전자 알고리즘 수행 중간의 CAG 모델을 나타내고, 그림 9는 최종적으로 찾은 CUG 모델과 CAG 모델을 나타낸다. 그림 8의 (a)는 그림 3의 (b) CAG 모델에서 응집도가 가장 낮은 클래스 Q를 재구조화한 CAG 모델이고, 그림 8의 (b)는 그림 8의 (a)에서 클래스 S를 재구조화한 후의 CAG 모델이다.

그림 9의 (a)와 (b)는 그림 8의 (b)에서 클래스 P와 Q를 재구조화한 CUG 모델과 CAG 모델이다. 그림 3의 모델과 그림 9의 모델을 비교하면 클래스 Q의 메소드 f3()과 속성 x가 클래스 Q보다 클래스 R과 정보 교환이 많기 때문에 클래스 R로 이동되었다. 그리고 클래스 P의 메소드 f1()과 클래스 Q의 메소드 f4()는 다른 메소드 또는 속성과 정보 교환이 없기 때문에 새로운 클래스 T를 만들어 그 클래스로 이동하였다. 클래스 S는 분해되어 메소드 f8()과 속성 m, n은 클래스 P로 이동하였고, 메소드 f7()은



(a) 클래스 Q 재구조화

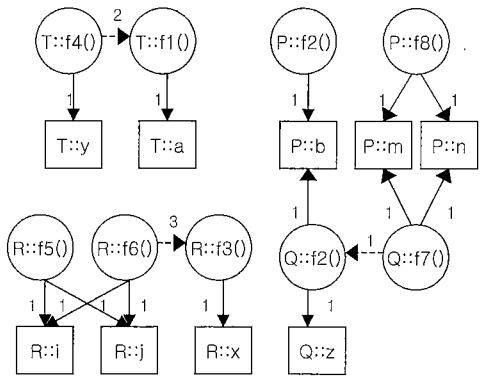


(b) 클래스 S 재구조화

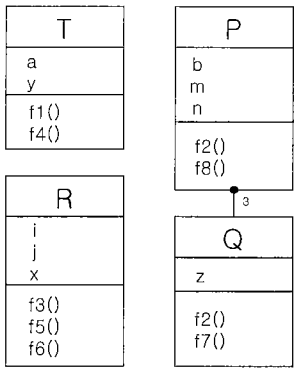
그림 8 재구조화 과정 중의 CAG 모델

클래스 P의 자식 클래스 Q로 이동하였다.

표 4는 초기 모델(그림 3)의 메트릭, 재구조화 과정중 모델(그림 8)의 메트릭, 그리고 재구조화된 모델(그림 9)의 메트릭을 보여준다. 표 4에서 보면 초기 설계의 *COH*, COP_{IH}, 그리고 COP_{IA}이 각각 2.65, 1.67, 4.39인데 재구조화된 후에는 각각 2.96, 1.81, 0으로서 전체적인 설계 메트릭 값이 많이 개선된 것을 볼 수 있다. 이렇게 개선된 이유는 첫 번째로 응집도가 낮은 클래스 Q와 P가 분해되어 메소드와 속성이 클래스 R과 T로 이동함으로써 응집도가 증가되고 상호작용 결합도가 감소되었기 때문이다. 두 번째로 클래스 S가 분해되어 클래스 P와 Q로 이동함으로써 상호작용 결합도가 감소되고 상속 결합도가 증가되었기



(a) CUG 모델



(b) CAG 모델

그림 9 재구조화된 추상화 모델

때문이다. 표 5는 초기 설계 모델과 재구조화된 설계 모델을 기존의 메트릭을 사용하여 측정한 결과로서 각 클래스의 메트릭 값을 합하여 전체 클래스 개수로 나누어 표시하였다. 이 표에서 재구조화 과정 전후의 설계 응집도와 결합도 메트릭 값을 보면, DIT와 LCOM[8]은 그대로이나

표 4 제안한 메트릭을 사용한 측정

설계	COH(D)	$COP_{III}(D)$	$COP_{IA}(D)$	No. of Classes
재구조화 전(그림 3)	2.65	1.67	4.39	4
그림 8의 (a)	2.76	1.67	1.67	4
그림 8의 (b)	1.76	1.67	0	3
재구조화 후(그림 9)	2.96	1.81	0	4

표 5 기존 메트릭을 사용한 측정

메트릭 / 설계	응집도		결합도			
	LCOM[8]	LCOM[9]	MPC[3]	DAC[3]	CBO[8]	DIT[8]
재구조화전 설계	1	1.25	0.75	0.5	1.5	0.25
재구조화후 설계	1	0.5	0	0	0.5	0.25

다른 설계 메트릭 값이 향상된 것을 알 수 있다. DIT는 클래스 P와 Q의 가상 함수 f2()때문에 변하지 않았다. 따라서 본 논문에서 제안한 메트릭과 기존의 메트릭이 유사한 결과를 나타내므로 본 재구조화 방법은 객체 지향 설계 품질을 향상시킬 것으로 기대된다.

5. 결론 및 향후 연구

본 논문에서는 메트릭을 이용하여 객체 지향 설계를 자동적으로 재구조화하는 방법을 기술하였다. 자동화를 위하여 먼저 객체 지향 설계를 가시적인 추상화 모델로 표시하고 이 모델로부터 설계를 정량적으로 측정할 수 있는 응집도와 결합도 메트릭을 정의한다. 이 가시적인 모델은 재구조화 과정을 쉽게 이해하는데 도움을 제공한다. 정의한 응집도와 결합도 메트릭을 이용하여 적합도 함수를 만들고 유전자 알고리즘을 적용하여 객체 지향 설계를 자동적으로 재구조화한다. 본 재구조화 방법은 다음과 같은 장점이 있다. 첫째, 본 방법은 기존 시스템의 설계 문제점을 개선하기 위하여 프로그램 소스로부터 정보를 추출하여 설계를 재구조화하기 위하여 적용되거나 또는 소프트웨어 설계 단계에서 모델링 도구로부터 정보를 추출하여 설계 품질을 높이기 위하여 적용될 수 있다. 둘째, 설계 재구조화 과정을 자동화함으로써 소프트웨어 개발 또는 유지 보수 단계에서 생산성을 향상시킬 수 있다. 마지막으로 정보 교환 행렬에 메소드 호출 횟수 정보를 추가하면 실행 시간 메트릭을 사용한 재구조화가 가능하므로 본 방법은 분산 환경에서 객체 지향 시스템을 동적으로 재구조화하기 위하여 적용될 수 있다.

향후에는 프로그램 언어 또는 모델링 도구에 독립적인 추상 객체 지향 설계 언어(abstract object-oriented design language)를 정의하고 모델링 도구 또는 프로그램으로부터 자동적으로 이 언어로 표현된 설계 정보를 추출하기 위한 방법과 구현을 지원하기 위하여 재구조화된 설계로부터 소스 코드를 생성하는 방법을 연구할 것이다. 그리고 본 연구를 규모가 큰 실제 시스템에 적용하여 실험적으로 그 유효성을 확인할 것이다.

참 고 문 헌

[1] R. Arnold, "Software Restructuring," *Proceedings of the IEEE*, Vol. 77, No. 4, pp. 607-617, 1989.

[2] S. R. Chidamber and C. F. Kemerer, "Towards a Metrics Suite for Object-Oriented Design," *In Proceedings of Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'91)*, SIGPLAN Notices, Vol. 26, No. 11, pp. 197-211, 1991.

[3] Li W, Henry S., "Object-Oriented Metrics That Predict Maintainability," *Journal of Systems and Software*, Vol. 23, pp. 111-122, 1993.

[4] L. Briand, P. Devanbu, and W. Melo, "An Investigation into Coupling Measures for C++," *In Proceedings of 19th International Conference on Software Engineering*, pp. 412-421, 1997.

[5] Y. S. Lee, B. S. Liang, S. F. Wu, and F. J. Wang, "Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow," *In Proceedings of International Conference on Software Quality*, 1995.

[6] S. Yacoub, H. Ammar, and T. Robinson, "Dynamic Metrics for Object Oriented Designs," *In Proceedings of 6th International Symposium on Software Metrics, 1999.*

[7] E. B. Allen and T. M. Khoshgoftaar, "Measuring Coupling and Cohesion: An Information-Theory Approach," *In Proceedings of International Symposium on Software Metrics*, 1999.

[8] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object-Oriented Design," *EEE Transactions on Software Engineering*, Vol. 20, No. 6, pp. 476-493, 1994.

[9] M. Hitz, B. Montazeri, "Measuring Coupling and Cohesion in Object-Oriented Systems," *n Proceedings of International Symposium on Applied Corporate Computing*, 1995.

[10] J. M. Bieman and B. K. Kang, "Cohesion and Reuse in an Object-Oriented System," *In Proceedings of ACM Symposium Software Reusability(SSR'95)*, pp. 259-262, 1995.

[11] H. S. Chae and Y. R. Kwon, "A Cohesion Measure for Classes in Object-Oriented Systems," *In Proceedings of 5th International Software Metrics Symposium*, pp. 158-166, 1998.

[12] L. C. Briand, S. Morasca, and V. R. Basili, "Defining and Validating Measures for Object-ased High-Level Design," *IEEE Transactions on Software Engineering*, Vol. 25, No. 5, pp. 722-743, 1999.

[13] L. C. Briand, J. W. Daly, and J. Wüst, "A Unified Framework for Cohesion Measurement in Object-Oriented Systems," *In Proceedings of 4th International Software Metrics Symposium*, pp. 43-53, 1997.

[14] L. C. Briand, J. W. Daly, and J. Wüst, "A Unified Framework for Coupling Measurement in Object-Oriented Systems," *IEEE Transactions on Software Engineering*, Vol. 25, No. 1, pp. 91-121, 1999.

[15] L. C. Briand, S. Morasca, and V. R. Basili, "Property-Based Software Engineering Measurement," *IEEE Transactions on Software Engineering*, Vol. 22, No. 1, pp. 68-86, 1996.

[16] S. C. Choi and W. Scacchi, "Extracting and Restructuring the Design of Large Systems," *IEEE Software*, Vol. 7, No. 1, pp. 66-71, 1990.

[17] Fernando Brito e Abreu, Goncalo Pereira, and Pedro Sousa, "A Coupling-Guided Cluster Analysis Approach to Reengineer the Modularity of Object-Oriented Systems," *In Proceedings of 4th European Conference on Software Maintenance and Reengineering*, March 2000.

[18] Deborah Tesch and Gary Klein, "Optimal Module Clustering in Program Organization," *In Proceedings of the Twenty-Fourth Annual Hawii International Conference on System Sciences*, Vol. 2, pp. 238-245, 1991.

[19] H. S. Kim, Y. R. Kwon, and I. S. Chung, "Restructuring programs through program slicing," *International Journal of Software Engineering and Knowledge Engineering*, Vol. 4, No. 3, pp. 349-368, 1994.

[20] B. K. Kang and J. M. Bieman, "A Quantitative Framework for Software Restructuring," *Journal of Software Maintenance*, Vol. 11, No. 4, pp. 245-284, Jul-Aug, 1999.

[21] van Rijsbergen, C. J., *Information retrieval*, 2d ed. London:Butterworths, 1979.

[22] D. E. Goldberg, *Genetic Alogrithms in Search, Optimization & Machine Learning*, Addison Wesley, 1989.

[23] W. Stevens, G. Myers, and L. Constantine, "Structured design," *IBM Systems Journal*, Vol. 13, No. 2, pp. 115-139, 1974.

[24] J. Banerjee, W. Kim, H. J. Kim, and H. Korth, "Semantics and Implementation of Schema Evolution in Object-Oriented Databases," *ACM SIGMOD*, 1987.

이 병 정

1990년 서울대학교 계산통계학과 학사. 1990년 ~ 1996년 현대전자(주) 소프트웨어연구소 주임연구원. 1998년 서울대학교 전산과학과 석사. 1998년 ~ 현재 서울대학교 전산과학과 박사과정. 관심분야는 소프트웨어 재사용, 컴포넌트 기반 소프트웨어 개발, 분산 객체 기술 등

우 치 수

1972년 서울대학교 공과대학 응용수학(공학사). 1977년 서울대학교 대학원 전산학(석사). 1982년 서울대학교 대학원 전산학(박사). 1978년 영국 라퍼러대학연구원. 1975년 ~ 1982년 울산대 전자계산학과 부교수. 1985년 미국 미시건대학 Post-Doc. 현재 서울대학교 전산과학과 교수. 관심분야는 소프트웨어 공학, 프로그래밍 언어

.