혼성 메트릭을 이용한 소프트웨어 개체 복잡도 정량화 기법

홍 의 석[†]·김 태 균[††]

요 약

소프트웨어 개발 기술이 발전하고 소프트웨어 정량화의 중요성이 커지면서 많은 메트릭들이 여러 시스템 개체의 정량화를 위해 제안되었다. 이들은 크게 스칼라 메트릭 형태나 메트릭 벡터 형태를 취한다. 최근에 몇몇 연구들에서 스칼라 메트릭의 조합 형태에서 오는 위험성을 지적하였지만 아직도 유용성 등의 큰 이점 때문에 많은 스칼라 메트릭들이 사용되고 있다. 본 논문은 기존 메트릭 연구들의 분석 과정을 통해 스칼라 메트릭 형태는 외부 복잡도에 가중을 둔 혼성 메트릭 형태가 가장 적당하다는 결론을 얻었으며 이를 토대로 개발 방법론과 개발 시스템 형태에 의존하지 않는 일반적인 혼성 복잡도 메트릭 제작 프레임워크를 제안한다. 제안 프레임워크는 구조적 방법론의 분석 단계와 객체지향 실시간 시스템 설계 단계의 정량화 프로젝트에 사용되었으며 두 프로젝트 모두 만족할만한 결과를 얻었다. 정량화 목적을 갖는 개발 집단은 제안 프레임워크를 이용하여 단시간 내에 여러 종류의 시스템 개체를 정량화할 수 있다.

Quantification Methods for Software Entity Complexity with Hybrid Metrics

Euy-Seok Hong[†]·Tae-Guun Kim[††]

ABSTRACT

As software technology is in progress and software quantification is getting more important, many metrics have been proposed to quantify a variety of system entities. These metrics can be classified into two different forms : scalar metric and metric vector. Though some recent studies pointed out the composition problem of the scalar metric form, many scalar metrics are successfully used in software development organizations due to their practical applications. In this paper, it is concluded that hybrid metric form weighting external complexity is most suitable for scalar metric form. With this concept, a general framework for hybrid metrics construction independent of the development methodologies and target system type is proposed. This framework was successfully used in two projects that quantify the analysis phase of the structured methodology and the design phase of the object oriented real-time system, respectively. Any organization can quantify system entities in a short time using this framework.

키워드 : 복잡도(Complexity), 혼성 메트릭(Hybrid metrics), 메트릭 프레임워크(Metric Framework)

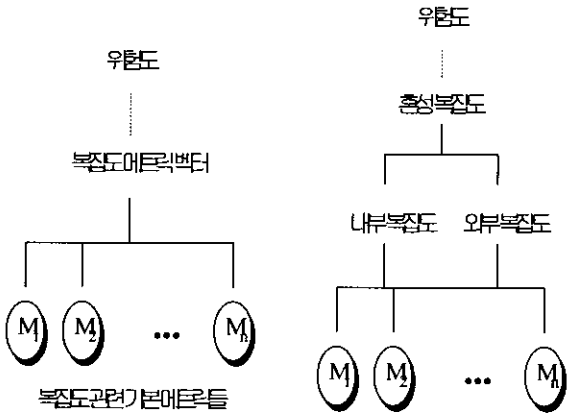
1. 서 론

소프트웨어 개발 기술이 발달함에 따라 각 개발 단계의 산물들과 프로세스들의 정량화 기술도 매우 중요해지고 있다. 많은 회사나 연구 기관들이 기존의 개발 프로세스를 측정 기반 소프트웨어 개발 프로세스(Measurement-based Software Development Process)로 변화 시켜 가고 있다. 이는 SEI의 CMM(Capability Maturity Model)[1]이나 ISO 9000-3[2]과 같은 개발 프로세스 개선 방법의 표준안들에서도 잘 나타나 있다. 프로세스 평가 기준인 여러 수준들 중 상위 수준으로 올라갈수록 잘 정의된 메트릭 집합을 이용한 개발 단계의 정량화와 이들을 이용한 프로세스 개선을 요구하기 때문이다. 이와 같은 정량화 요구에 부응하기 위해 수많은 메트릭들이 제안되고 사용되어 왔다.

소프트웨어 산물 메트릭이란 소프트웨어 시스템 내부를 구성하는 개체들을 정량화하기 위한 수단이며, 개체 정량화를 위해 크게 두가지 방법이 사용된다. (그림 1)은 개체 정량화의 두가지 방법을 사용하여 소프트웨어 품질 인자들 중 하나인 위험도(criticality)를 예측하는 기법을 나타낸다. 이와 같은 정량화 방법은 품질 인자를 복잡도 같은 관련된 부특성들로 나누고 각 부특성들을 관련 메트릭들로 측정한다는 점에서 McCall의 품질 모델 [3]이나 ISO/IEC 9126-1 품질 모델 [4]과 유사하다. 비록 위험도는 ISO/IEC 9126-1 모델에 정의된 품질 인자는 아니지만 최근에 중요한 소프트웨어 품질 인자로 인식되고있다. 본 논문에서 사용하는

† 정 회 원 : 안양대학교 디지털미디어학부 교수
†† 정 회 원 : 부산외국어대학교 컴퓨터공학과 교수
논문접수 : 2000년 4월 4일, 심사완료 : 2001년 5월 8일

정량화란 대상이 소프트웨어가 아니라 소프트웨어 설계 명세 또는 소프트웨어 코드에서 소프트웨어 시스템을 구성하는 하나의 개체이다.



(그림 1) 두가지 개체 정량화 방법

첫 번째 방법은 개체 E의 복잡도 관련 기본 메트릭들을 정의하고 이들로 구성된 복잡도 메트릭 벡터를 만드는 것이다. E의 벡터값이 측정되면 사람은 이를 해석하여 위험도를 추정한다. 두 번째 방법은 역시 개체 E의 기본 메트릭들을 정의하고 이들이 개체 복잡도의 어느 부분에 영향을 미치는가를 결정해 이들을 조합하여 E를 하나의 메트릭 값으로 정량화 하는 것이다. E의 기본 메트릭들로부터 E의 내부 복잡도와 외부 복잡도 메트릭을 정의하고 이를 이용하여 혼성 메트릭 형태로 E의 복잡도를 정의한다. 결국 두 가지 방법은 품질 인자를 메트릭 벡터 형태 또는 하나의 스칼라 값으로 정량화 하는 것이다.

90년대 중반부터 많은 메트릭 이론가들이 첫 번째 방법을 지지하고 두 번째 방법 즉 기본 메트릭들의 조합으로 하나의 메트릭을 만드는 조합 메트릭 방법을 비판하였다[5, 6]. 왜냐하면 여러 분포를 따르는(다른 성질을 가지는) 메트릭들을 조합하면 원래의 각 메트릭이 지닌 성질들을 잃어버릴 가능성이 크다는 것 때문이다. 하지만 현재 소프트웨어 개발 조직에서 많이 쓰이고 있는 메트릭들은 조합 메트릭 형태가 많다. 이는 메트릭 벡터는 원소의 수가 많을 때 해석과 이해의 부담이 크기 때문이며 이미 많이 알려진 메트릭들이 조합 메트릭 형태를 지니고 있는 경우[7]가 많기 때문이다. 또한 정량화의 역할 중 매우 중요한 비중을 차지하는 개체의 품질 비교 작업에서도 스칼라 메트릭을 사용하는 것이 매우 용이하다.

많은 혼성 메트릭들이 제안되었지만 그들은 각기 다른 조합 메트릭 형태로 제안되었다. 그러므로 본 논문에서는 소프트웨어 개발 단계 산물들의 특성들을 이용해 즉시 해당 개체의 혼성 복잡도 메트릭을 제작할 수 있는 프레임워크를 제안한다. 제안 프레임워크는 구조적 방법론이나 객체지향 방법론 또는 실시간 시스템 등 제작 시스템의 특성이나 개발 방법에 의존하지 않는 일반적인 프레임워크이며, 프레임워크에 의해 제작된 설계 단계의 복잡도 메트릭들은 복잡도와 밀접한 연관이 있다고 알려진 유지보수성, 위험도 등의 품질 인자 예측에 사용 가능하다.

2장에서는 기존 여러 복잡도 메트릭들을 분석하여 복잡도 개념을 분류하고 3장에서는 혼성 복잡도 메트릭 프레임워크를 제안한다. 4장에서는 제안 프레임워크를 이용하는 두가지 사례 연구들을 기술하며, 5장에서 결론과 향후 연구 과제에 대해 기술한다.

2. 복잡도 메트릭

설계 메트릭이란 소프트웨어 설계 명세들을 이용하는 메트릭이다. 이는 코딩이나 유지 보수 단계의 산물로 얻어지는 코드 메트릭과는 구별된다. 복잡도 메트릭 연구로 대변되는 대부분의 산물 메트릭(product metrics)에 관한 연구는 70년대 초부터 시작되었으며 주로 코드 메트릭에 관련된 연구였다. 그러나 80년대 초반부터 소프트웨어 생명 주기 초기 단계에서 측정할 수 있는 메트릭 연구가 본격화되었다[7].

80년대 중반부터 소프트웨어 메트릭과 개발 자료들(에러들, 코딩 시간 등)과의 연관성에 대한 실험 연구들이 많이 진행되었다. Rombach는 이들 실험 결과들을 정리하였는데 이는 다음과 같다[8].

- 구조 설계(Architectural design) 결과와 유지보수성은 밀접한 연관성이 있다.
- 구조 설계의 특성은 설계 메트릭(주로 모듈 간 관계 측정)으로 나타낼 수 있다.
- 알고리즘적 메트릭(모듈 내부 성질 측정 : McCabe, Halstead 메트릭과 같은 코드 메트릭)과 유지보수성과의 연관성은 낮다.
- 위의 두 형태를 혼합한 혼성 메트릭과 유지보수성은 밀접한 연관성이 있다.

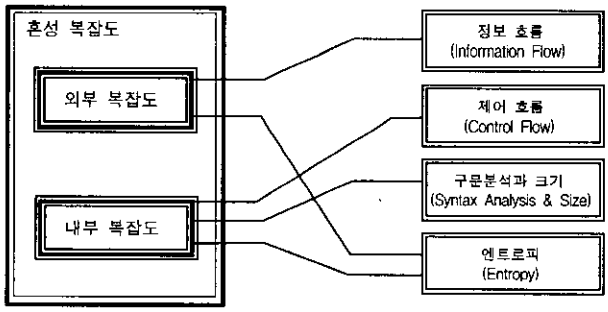
이와 같은 결과는 스칼라 형태의 복잡도 메트릭을 만들 때 알고리즘적 메트릭으로 측정 가능한 개체 내부의 복잡성과 개체와 개체간의 관계와 같은 개체 외부의 작용으로 생기는 외부 복잡성을 함께 고려한 혼성 메트릭 형태가 가장 좋은 결과를 보인다는 것을 의미한다. 또한 유지보수성 이외의 복잡도와 관련 있는 소프트웨어 품질 인자들에도 코드 메트릭보다 설계 메트릭이 유용하게 사용될 수 있다는 것을 암시한다. 복잡도 메트릭과 가장 연관성이 높은 품질 인자는 유지보수성이라 알려져 있다. 복잡도라는 것은 사람이 소프트웨어를 대하는 심리적 복잡도이며[9], 이는 소프트웨어 이해용이성(Understandability)과 직접적인 연관이 있기 때문이다.

2.1 복잡도 메트릭 분류

기존의 복잡도 메트릭 연구들은 두가지 관점에서 분류할 수 있다. 첫 번째는 시스템을 구성하는 모듈의 내부와 외부 관점에 의한 분류로서 모듈의 내부 요소들에 의한 내부 복잡도 메트릭, 모듈과 그의 환경과의 상호 작용에 의한 외부 복잡도 메트릭, 그들을 혼합한 혼성 복잡도 메트릭으로 분류하는 것이다. 두 번째는 복잡도에 영향을 미치는 여러 요인들 중 어떤 것에 치중했는가에 의해 분류할 수 있다. 다시 말하면 어떤 요인을 기준으로 하여 복잡도를 정의하였는가에 의한 분류이다. 이는 제어 흐름에 기초한 메트릭, 정보 흐름에 기초한 메트릭, 구문 분석과 크기에 기초한 메트릭, Shannon의 정보이론[10]에 기초한 엔트로피 사용 메트릭으로 나뉜다.

복잡도 메트릭의 두 분류간의 관계는 (그림 2)와 같다. (그림 2)의 양쪽 그룹은 앞에서 기술한 복잡도 메트릭의 두 분류라고 할 수 있지만, 왼쪽 그룹에 속하는 복잡도의 관점들을 오른쪽 그룹의 방법으로 측정할 수 있다는 해석도 가능하다. 예를 들면 외부 복잡도는 정보 흐름이나 엔트로피 방법을 사용하여 측정 가능하다.

외부 복잡도는 모듈과 시스템 환경 사이의 상호 작용에 의해 발생하는 것으로 환경은 다른 모듈들과 전역(공유) 변수들의 집합으로 구성된다. 모듈과 다른 모듈과의 상호 작용은 직접적인 호출에 의한 자료의 전달이나 전역 변수들을 공유함으로써 일어난다. 그러므로 외부 복잡도는 모듈간의 결합도와 밀접한 연관이 있다. 외부 복잡도를 측정하기 위한 메트릭은 정보 흐름 메트릭과 엔트로피 이용 메트릭이 있다. 이들은 모두 모듈간에 주고받는 정보의 양이 많을수록 모듈간의 결합도가 높아져 모듈의 복잡도가 커짐을 가정한다.



(그림 2) 복잡도 메트릭 분류

본 논문에서 언급하는 외부 복잡도와 내부 복잡도는 ISO/IEC 9126에서 정의한 외부 품질 및 내부 품질 그리고 외부 메트릭 및 내부 메트릭과는 다르다. 근본적인 차이는 앞에서도 기술하였듯이 정량화 대상의 차이이다. 본 논문에서는 정량화 대상이 시스템을 구성하는 개체이므로 외부 및 내부라는 표현은 개체의 외부와 내부가 되는 반면 ISO/ IEC 9126에서 언급한 외부와 내부는 소프트웨어의 외부와 내부를 의미한다. 그러므로 본 논문에서 언급하는 시스템 개체의 외부 복잡도 메트릭들과 내부 복잡도 메트릭들은 모두 ISO/IEC 9126-3에 정의된 내부 메트릭들의 범주에 속한다. 물론 ISO/IEC 9126 모델은 ISO/IEC 9126-3에서 예로 든 기본 메트릭들을 조합하여 하나의 스칼라 메트릭으로 만드는 기법에 대해서는 언급하지 않았다.

가장 대표적인 정보 흐름 메트릭은 Henry 등의 메트릭이다[7]. 그들은 정보 흐름을 지역 흐름과 전역 흐름으로 분류하였으며 이들 중 입력 흐름에 속하는 양을 fan-in, 출력 흐름에 속하는 양을 fan-out으로 정의하였다. 그들의 초기 메트릭은 혼성 메트릭 형태인 $LOC \cdot (fan\text{-}in \cdot fan\text{-}out)^2$이며 후에 LOC 부분을 내부 복잡도를 나타내는 임의의 코드 메트릭으로 수정하여 정의하였다[11]. 이와 같은 형태는 전체 시스템 관점에서는 한 모듈의 fan-in이 다른 모듈의 fan-out이 되는 중복된 의미가 있으므로 Card 등은 fan-out에 중점을 둔 새로운 복잡도 메트릭을 정의하였다[12]. 이는 Henry 메트릭의 또 다른 문제점인 메트릭 내의 한 요소가 0이 되면 메트릭 값이 0이 된다는 것도 해결하지만 모듈간의 결합도 요소를 저 평가함으로써 메트릭이 갖추어야 할 변별 능력(Discriminative power)을 약화시킨다.

Henry 메트릭은 입출력 정보 흐름 형태를 구별하지 않았지만 Zage 등은 모듈간 호출 흐름의 입출력량과 데이터 흐름의 입출력량을 곱하여 그들의 가중합 형태로 외부 복잡도를 정의하였다[13]. 입출력 양을 곱하는 것이 카테지안 곱(데카르트 곱)의 양에 기초한 것이라 보면 동종의 정보량의 곱을 고려하는 것이 종류를 고려치 않는 것보다 더 의미가 있다. Lew 등은 모듈간에 주고받는 메시지에 함축된 정보량을 엔트로피를 사용하여 정의하였다[14]. 그들은 메시지 형태를 상태(제어)와 데이터 메시지로 나누고 그들이 갖는 정보량들과 자료 구조가 갖는 정보량들의 합으로 외부 복잡도를 정의하였다.

내부 복잡도는 모듈 내부만의 수행이나 크기 또는 내부 요소간의 상호 작용들에 의해 발생되는 복잡도이기 때문에 모듈의 코드나 상세 설계를 필요로 한다. 제어 흐름 또는 구문 분석과 크기에 기초한 메트릭은 내부 복잡도 측정에 사용된다. 전자의 예로는 McCabe의 Cyclomatic Complexity와 그의 변형들[15]이 있고, 후자의 예는 LOC 또는 Halstead의 Software Science[16] 그리고 Albrecht의 Function Point 등이 있다. Harrison은 모듈 내 연산자들의 분포를 엔트로피 이론에 적용하여 모듈의 코드가 가지는 정보량을 측정하였다[17]. 이들의 문제점은 구조 설계 결과 정보를 이용하여서는 측정할 수 없다는 것이다.

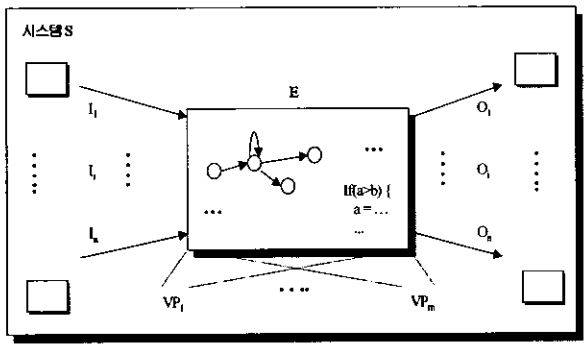
3. 혼성 복잡도 메트릭 제작 프레임워크

Henry와 다른 몇몇 연구자들은 외부 복잡도와 내부 복잡

도는 모듈 복잡도에 대한 서로 다른 관점이므로 이들을 혼합한 혼성 메트릭 형태가 바람직하며, 외부 복잡도가 내부 복잡도보다 모듈 복잡도에 미치는 영향이 크다고 주장하였다[7, 11, 18]. 이는 다른 연구자들에 의해 부분적으로 부정되었지만[19], 앞에서 기술한 Rombach의 실험 결과 역시 Henry의 주장을 뒷받침한다. 그러므로 본 논문에서 제안하는 조합 메트릭 제작 프레임워크는 외부 복잡도에 높은 가중치를 둔 혼성 복잡도 메트릭 형태를 취한다.

복잡도의 다른 두 관점을 혼합하는 것은 매우 주의를 요한다. 왜냐하면 두 메트릭들을 혼합한 조합 메트릭은 두 메트릭들의 성질들을 모두 잃을 가능성이 있기 때문이다. 혼합 형태는 가중곱 형태와 가중합 형태를 사용한다. 전자는 ***내부복잡도 · 외부복잡도***[2] 형태이고 후자는 w_i ***· 내부복잡도***+w_e ***· 외부복잡도*** 형태이다. 전자는 제곱을 사용하여 외부 복잡도에 가중을 두었고, 후자는 외부 복잡도의 가중치인 w_e를 내부 복잡도의 가중치인 w_i보다 크게 함으로써 외부 복잡도에 가중을 두었다. 가중곱 형태의 메트릭 값 분포는 가중합 형태보다 산포도가 크므로 메트릭의 변별 능력은 높게 하지만 이상점이 생길 가능성이 크며 가중합 형태는 가중치를 결정해야 하는 어려움이 있다.

앞의 여러 메트릭 연구들의 분석을 통해 다음과 같은 혼성 복잡도 메트릭 제작 프레임워크를 정의할 수 있다. 이 프레임워크는 시스템 내에 존재하는 기능 개체 E의 혼성 복잡도 메트릭 구축 방법이다. (그림 3)은 시스템 S를 구성하는 일반적인 기능 개체 E를 두가지 큰 관점으로 나타내었다. 이 중 하나는 E의 내부 기능에 대한 여러 가지 관점들로 이루어지고, 다른 하나는 E와 E 외부의 다른 시스템 개체들과의 상호 작용으로 이루어진다.



(그림 3) 시스템 내 기능 개체의 추상화

(그림 3)의 시스템 S는 시스템 개발의 각 단계에서 보여지는 시스템 개발 산물이다. 예를 들면 시스템 분석 단계에서는 시스템 설계에 필요한 여러 모듈들이나 구체적인 모듈간의 입출력, 그리고 각 모듈의 내부 기능이 정의되지 않는다. 단지 시스템이 어떤 일을 수행하는가에 대한 자료흐름도와 자료사전 정도가 정의될 뿐이다. 그러므로 시스템 분석 단계와 시스템 설계 단계에서의 (그림 3)의 모양은 달라진다. 이는 시스템 설계 단계와 시스템 구현 단계에서도 마찬가지로 적용된다. 다시 말하면 (그림 3)은 시스템 개발 중 한 단계의 산물로 본 시스템 S에 대한 표현이다. 각 단계마다 시스템에 대한 관점이 달라지며 정량화 하려는 개체의 종류도 달라진다. 예를 들면 분석 단계에서는 자료흐름도의 프로세스, 설계 단계에서는 모듈, 구현 단계에서는 함수 등이 정량화 하려는 개체가 될 수 있다.

- E : 시스템 S를 구성하는 임의의 기능 개체
- I_i, O_i : 같은 종류의(i번째의) 입출력 정보 집합 ($1 \le i \le n$)

예를 들어 E가 모듈화 프로그래밍 언어의 한 모듈이라면 (I_i, O_i)[1]는 (E의 외부에서 E를 호출하는 모듈들의 집합, E가 호출하는 외부 모듈들의 집합), (호출에 의해 E로 들어오는 데이터 집합, 호출에 의해 E에서 나가는 데이터 집합), (E에서 읽는 전역 변수 집합, E에서 값을 변경하는 전역 변수 집합) 등이다. 만약 E가 객체지향 시스템의 클래스라면 (I_i, O_i)는 (입력 메시지 집합, 출력 메시지 집합), (입력 데이터 집합, 출력 데이터 집합), (상속 구조에서 부모 클래스 집합, 상속 구조에서 자식 클래스 집합) 등이다.

- VP_i : i번째 관점에 의한 E 내부의 정보 집합 ($1 \le i \le m$)

E의 내부 구조나 수행 원리는 여러 관점으로 볼 수 있다. 예를 들면 제어 흐름들의 집합으로 볼 수도 있고, 문법에 의한 구문 분석에 의한 정보 집합 또는 여러 수행 상태들과 그들 간의 전이의 집합으로 볼 수도 있다.

- IC(E) : E의 내부 복잡도

$$IC(E) = f(\Sigma |VP_i|),\ f : Z+ \rightarrow R+\text{인 증가함수} \qquad (1)$$

Z+ = {x | x∈Z, x≥0} R+ = {x | x∈R, x≥0}[2]

내부 복잡도는 각 관점의 정보들의 크기들을 더한 값이 증가함에 따라 함께 증가하는 함수값으로 정의하였다. $|VP_i|$를 더할 때의 문제점은 각 VP_i가 서로 의존적인 관계에 있는 경우이다. 예를 들면 한 클래스의 클래스 계층도에서 위치값과 클래스 특성수는 하위 클래스로 갈수록 더욱 많아지므로 서로 의존 관계에 있다. 그러므로 이러한 경우는 의존 관계를 줄이는 새로운 통합 관점 집합을 정의하는 것이 바람직하다.

- EC(E) : E의 외부 복잡도

$$EC(E) = g(\Sigma((|I_i|+c) \cdot (|O_i|+c))),$$
$$g : Z+ \rightarrow R+\text{인 증가함수},\ c \in Z+ \qquad (2)$$

외부 복잡도는 같은 종류의 입출력 정보의 카테지안 곱들을 더한 값이 증가함에 따라 증가하는 함수값으로 정의

1) 편의상 집합 A와 B의 쌍을 (A, B)라 나타낸다.
2) Z : 모든 정수의 집합, R : 모든 실수의 집합

하였다. 입출력 정보 부분에 상수 c를 더한 것은 입력이나 출력의 정보 중 하나가 0이 되는 경우를 해결하기 위해서이다. 만약 모듈의 입력이 여러 개인데 모듈 내부 연산만 이루어지고 출력이 없다면 출력정보의 수가 0이 되므로 입출력 정보수의 곱은 0이 된다.

- C(E) : E의 혼성 복잡도($C1$: 가중합 형태, $C2$: 가중곱 형태)

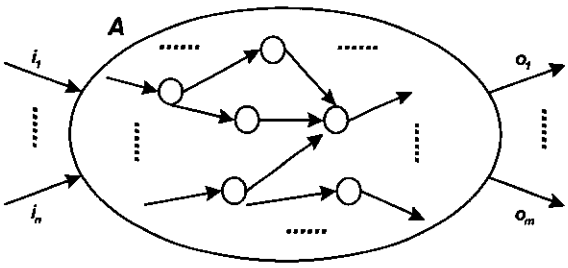
$$C1(\mathrm{E}) = w_i \cdot IC(\mathrm{E}) + w_e \cdot EC(\mathrm{E}), \quad (w_i < w_e) \qquad (3)$$

$$C2(\mathrm{E}) = IC(\mathrm{E}) \cdot EC(\mathrm{E})^2 \qquad (4)$$

4. 프레임워크 사용 예

4.1 자료흐름도에서 프로세스 복잡도

제안한 프레임워크는 구조적 개발 방법론을 지원하는 통합 CASE 도구인 ADVISE의 서브 시스템인 ADVISE/DFD2SC 구현에 사용되었으며 요구분석 단계의 산물들인 자료흐름도와 자료사전을 이용하여 구조도를 자동 생성하는 변환분석의 중심변환 선정 알고리즘에 자료흐름도에서의 기능 단위인 프로세스의 복잡도 메트릭을 정의 이용하였다[20]. 중심변환이란 자료흐름도를 구성하는 프로세스들 중 가장 중심 기능을 하는 프로세스 집합을 의미한다.



(그림 4) 자료흐름도 프로세스 구성도

이 연구의 기본 생각은 한 프로세스의 복잡도는 처리한 자료흐름의 양(연결 복잡도)과 프로세스를 형성하는 근원 프로세스들의 수(내부 복잡도)에 의존하고, 복잡도가 높은 프로세스들이 중심변환이 될 가능성이 높다는 것이다. 프로세스의 연결 복잡도 즉 외부복잡도는 프로세스의 기능에 영향받는 자료흐름들의 의미적인 변화량으로 측정한다. 자료흐름의 이름은 자료사전에 여러 자료 항목들의 조합으로 정의되어 있으며 각 자료흐름간의 의미적 유사성은 그들을 정의하는 자료 항목들을 비교함으로써 정량화 한다. (그림 4)는 프로세스 A의 입출력 자료흐름들($i_1 \cdots i_n$, $o_1 \cdots o_m$)과 A의 후손 자료흐름도들을 통합한 자료흐름도를 나타낸다. 즉 A의 내부에 그려진 자료흐름도의 프로세스들이 근원 프로세스들이다.

(그림 4)의 프로세스 A의 정량화 결과는 다음과 같다.

- $C_B = (|(\bigcup_{k=1}^{n} S_{i_k} - \bigcup_{k=1}^{m} S_{o_k})| + 1) \times (|(\bigcup_{k=1}^{m} S_{o_k} - \bigcup_{k=1}^{n} S_{i_k})| + 1)$
- C_I = 근원 프로세스의 수(근원 프로세스의 C_I = 1)
- $C_P = w_B \cdot C_B + w_I \cdot C_I$

S_p = {x|x는 자료흐름 p의 자료사전 정의에 속하는 자료항목}이며, C_B는 연결 복잡도, C_I는 내부 복잡도, C_P는 프로세스 복잡도이다. C_B는 출력 부분 자료흐름들의 자료사전에 정의된 자료 항목들과 일치하지 않는 입력 부분의 자료 항목의 개수에 1을 더한 값과 같은 조건을 갖춘 출력 부분의 자료 항목의 개수에 1을 더한 값을 곱한 값이다. 곱셈을 사용함으로써 입력 부분의 사라진 자료 항목들과 출력 부분의 생성된 자료 항목들의 카테지안 곱의 수와 비례하는 값이 되게 하였다.

이와 같은 정량화 결과를 3장의 프레임워크에 맞추어 살펴보면 다음과 같다.

< 자료흐름도 내부의 프로세스 E의 정량화 >

VP_1 = {x | x는 E의 근원 프로세스 }

I_1 = {x | x는 출력 부분 자료흐름들의 자료사전에 정의된 자료 항목들과 일치하지 않는 입력 부분의 자료 항목}

O_1 = {x | x는 입력 부분 자료흐름들의 자료사전에 정의된 자료 항목들과 일치하지 않는 출력 부분의 자료 항목}

. $IC(E) = |VP_1|$, f(x) = x

$EC(E) = |I_1+1||O_1+1|$, g(x) = x

C1(E) = $w_i \cdot IC(\mathrm{E}) + w_e \cdot EC(\mathrm{E})$, $(w_i, w_e) = (0.2, 0.8)$

개체 내부의 복잡도에 대한 관점은 근원 프로세스의 구성이라는 한가지 관점만 사용하였지만 근원 프로세스가 단순히 프로세스 내부를 구성하는 바로 밑 단계의 프로세스를 의미하는 것이 아니라 전체 프로세스 계층 구조에서 가장 하위 계층의 프로세스를 의미하기 때문에 실제로는 정량화 하려는 프로세스의 계층에 대한 관점도 VP1에는 속해있다. 입출력 정보 역시 단순한 입출력 자료흐름들의 수로 측정하지 않고 보다 정확한 입출력 정보인 자료사전에 근거한 정보를 사용하였으며 이는 자료흐름수와 밀접한 연관 관계가 있기 때문에 자료흐름에 대한 입출력 정보를 정량화 결과에 포함시키지 않았다. 자료흐름도의 프로세스 정량화 사례 연구 결과는 3장의 프레임워크를 만족한다.

4.1.1 가중합 형태의 가중치 결정

w_I와 w_B 값을 정하기 위해 많이 알려진 구조적 분석 관련 서적[21, 22, 23, 24]에서 13개의 자료흐름도를 선택해 복잡도에 의한 중심변환 선정과 사람의 선정 결과를 비교하였다. <표 1>은 실험 결과이다. 각 자료흐름도에 대해 사람이 적당한 중심변환들을 선정하였으며 가급적이면 선정수를 전체 프로세스 수의 15% 정도에 맞추었다. 또한 각 자료흐름도에 대해 선정된 수만큼의 복잡도가 큰 프로세스

들을 선정하였다. 적중률은 이들 13개의 자료흐름도에서 선정된 총 중심변환 수에 대해 복잡도에 의해 선정한 프로세스들의 일치수의 비율이다.

〈표 1〉 가중합 메트릭 형태의 가중치 결정

w_I, w_B	선정 중심변환 수 (ACT)	일치하는 수 (SCT)	적중률 (SCT/ACT)
0.1, 0.9	207	165	79.7%
0.2, 0.8	207	184	88.8%
0.3, 0.7	207	159	76.8%
0.4, 0.6	207	141	68.1%

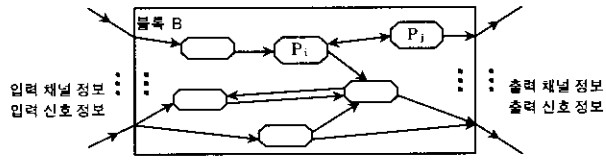
실험 결과 w_I와 w_B가 각각 0.2, 0.8인 경우에 가중합 메트릭의 효율이 가장 좋았다. 제안 프레임워크를 사용한 다른 사례연구에서도 식 (3)의 w_i와 w_e를 0.2와 0.8로 하여 만족할만한 결과를 얻었다. 하지만 자료흐름도에서 중심변환 선정의 예가 일반적인 것이 아니고 모든 예에 적용할 수 있는 가장 적당한 가중치의 존재 여부도 불투명하므로 가중치 결정은 보다 많은 다른 분야의 예들을 통해 이루어져야 한다.

4.2 SDL 설계 명세에서 블록 복잡도

SDL은 ITU-T Z.100 표준안으로 널리 사용되고 있는 객체지향 실시간 시스템 명세 언어이다. [25]는 SDL 시스템에서 정량화할 대상을 구조 설계 단계의 블록, 상세 설계 단계의 블록과 프로세스로 나누고 세가지 개체형에 대한 정량화 방법을 제안하였다. 정량화는 (그림 1)에서 언급한 메트릭 벡터 형태와 스칼라 메트릭 형태를 모두 제안하였으며 본 논문에서 제안하는 프레임워크는 스칼라 메트릭을 제작하는 데 사용되었다. 본 절에서는 구조 설계 단계의 블록의 혼성 복잡도 메트릭에 대해 프레임워크 사용 예를 살펴본다.

SDL 메트릭들은 SDL 설계 단계와 각 단계의 산물들을 모델화한 구조화된 SDL 모델 [25]로부터 정의되며, FBID (Flat BID)는 구조 설계 단계에서 시스템을 가장 자세한 단위들로 분할하였을 때 전체 시스템을 나타내는 BID(Block Interaction Diagram)이고 FPD(Flat PD)는 상세 설계 단계에서 시스템을 가장 자세하게 나타낸 PD(Process Diagram)이다[3].

구조 설계 단계는 시스템의 블록 구조와 끝단 블록들의 BID가 결정되는 단계이므로 블록간의 상호 작용은 채널과 신호에 의해서만 이루어지고 각 프로세스간에 주고받는 데이터는 정의되지 않는다. 구조 설계 단계에서 프로세스와 프로세스 외부와의 상호 작용은 라우트들과 신호들에 의해 정의할 수 있으나 프로세스의 내부에 대한 정보는 정의되지 않는다. 그러므로 구조 설계 단계에서 프로세스는 적절한 관심 개체가 아니다. (그림 5)는 블록 B를 나타내는 사각형 심볼 안에 그의 상세 정보인 FBID를 나타낸 것이다. 블록 B에 대한 (그림 5) 표현은 (그림 3)에서 일반적인 시스템 내의 개체 E의 한 예이다.



(그림 5) SDL 시스템에서 구조 설계 단계의 블록

(그림 5)의 SDL 블록 B의 정량화 결과는 다음과 같다.

- $AIBC = BP + BS + BR$ (*BP, BS, BR* : number of Processes /Routes/Signals of a Block)
- $AEBC = IBC \cdot OBC + IBS \cdot OBS$ (*IBC, OBC, IBS, OBS* : number of Input/Output Channels/Signals of a Block)
- $ABC1 = w_i \cdot AIBC + w_e \cdot AEBC$
- $ABC2 = AIBC \cdot AEBC^2$

이와 같은 정량화 결과를 3장의 프레임워크에 맞추어 살펴보면 다음과 같다.

< 구조설계 단계의 SDL 블록 E의 정량화 >

VP_1 = {x | x는 E의 FBID를 구성하는 프로세스 }
VP_2 = {x | x는 E의 FBID를 구성하는 라우트 }
VP_3 = {x | x는 E의 FBID를 구성하는 신호 }
I_1 = {x | x는 E의 입력채널} O_1 = {x | x는 E의 출력채널}
I_2 = {x | x는 E의 입력신호} O_2 = {x | x는 E의 출력신호}
$IC(E) = \sum |VP_i|$, f(x) = x
$EC(E) = \sum (|I_i| \cdot |O_i|)$, $g(x) = \sqrt{x}$
$C1(E) = w_i \cdot IC(E) + w_e \cdot EC(E)$, $(w_i, w_e) = (0.2, 0.8)$
$C2(E) = IC(E) \cdot EC(E)^2$

*AEBC*는 최근에 SDL 메트릭 집합을 개선하는 연구에서 차원 분석의 검증 조건 [6]을 만족시키기 위해 $\sqrt{IBC \times OBC + IBS \times OBS}$로 변형되었다[26]. 하지만 제곱근 함수 역시 증가 함수이므로 프레임워크를 만족한다. 사례연구에서 ABC1과 ABC2의 유용성을 보이기 위해 설계 단계의 결과를 이용해 위험도를 예측하는 실험을 하였으며 기존에 알려진 위험도 예측 모델들 중 매우 우수하다고 평가된 역전파 신경망 모델과 비교한 결과 거의 근접하는 예측률을 얻었다[27]. 이 실험에서도 역시 ABC1를 위해 가중치 0.2, 0.8을 사용하였다. 이는 메트릭 벡터를 입력으로 하고 모델 훈련을 위해 훈련 데이터 집합을 필요로 하는 역전파 신경망 모델의 결정적

3) 본 논문에서 SDL 메트릭 정의에 관련된 용어들과 배경 이론에 대한 자세한 설명은 [25]를 참조하기 바란다.

인 문제점을 고려할 때 매우 고무적인 결과였다. 이 연구에서 하였던 또 다른 실험은 ABC1과 ABC2의 상관성 실험이었다. 실험 결과 가중합과 가중곱 메트릭값은 순서적인 선택 목적(ordinal purpose)으로 사용 시에는 매우 높은 상관관계가 있음을 알아내었다. 하지만 이 결과 역시 정량화를 위해 가중합과 가중곱 중 한가지만 사용하면 된다는 것을 의미하지는 않는다. 두 조합 메트릭을 구성하는 기본 메트릭들의 변화에 따라 두 조합 메트릭들의 값이 같은 형태로 변화하지는 않는다. 이것은 합과 곱의 메트릭 형태에서 오는 근본적인 차이점이다. 그러므로 시스템 개체 정량화 시 두 형태를 모두 사용하여 결론을 내리는 것이 바람직하다.

SDL의 세가지 형태의 개체를 정량화하는 메트릭들은 복잡도 메트릭들이 갖추어야하는 여러 성질들에 기반해 분석적으로도 검증되었다[26]. 검증에 사용된 메트릭들은 일반적인 형태를 취했으며 이는 3장의 제안 프레임워크를 구성하는 메트릭들이다. 그러므로 제안 프레임워크는 분석적 검증과 함께 본 논문에서 기술한 두 개의 정량화 프로젝트를 통한 실험적 검증을 마친 상태이다.

5. 결론 및 향후 연구

수많은 메트릭들이 여러 시스템 개체의 정량화를 위하여 제안되어 왔으며 그들은 메트릭 벡터 형태나 스칼라 메트릭 형태를 취한다. 최근에 스칼라 메트릭 형태의 위험성을 지적하는 연구들이 많이 발표되고 있지만 아직도 그 유용성 때문에 스칼라 메트릭은 많이 사용되고 있다. 본 논문은 기존 메트릭 연구들의 분석을 통해 스칼라 메트릭 형태는 외부 복잡도에 가중을 둔 혼성 메트릭 형태가 가장 적당하다는 결론을 냈으며 이를 토대로 개발 방법론과 개발 대상 시스템 형태에 의존하지 않는 일반적인 혼성 복잡도 메트릭 제작 프레임워크를 제안하였다. 제안 프레임워크는 두 정량화 프로젝트에서 사용되었으며 만족할만한 실험 결과를 보였다. 또한 분석적 검증도 수행되었다. 정량화 목적을 갖는 어떠한 개발 집단에서도 제안 프레임워크를 이용하여 단시간 내에 여러 시스템 개체의 정량화를 수행할 수 있다.

향후 연구는 보다 많은 정량화 프로젝트에 본 프레임워크를 사용하여 일반적인 가중합 형태의 가중치 존재 여부와 그 값을 찾아내는 것이고, 가중합 형태와 가중곱 형태가 순서적인 선택 목적으로 교환 사용 가능한지에 대한 일반적인 결론을 내리는 것이다.

참 고 문 헌

[1] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber, Capability Maturity Model for Software, Version 1.1, CMU/SEI-93-TR-24, 1993.

[2] International Organization for Standardization, *Quality Management and Quality System Elements(Part 3) Guidelines for Development, Supply and Maintenance of Software*, ISO 9000-3.

[3] J. McCall, P. Richards, and G. Walters, "Factors in Software Quality," *three volumes, NTIS AD-A049-014, 015, 055*, 1997.

[4] ISO/IEC JTC1/SC7/WG6, ISO/IEC 9126 : Information Technology - *Software quality characteristics and metrics*

[5] N. Fenton, "Software Measurement : A Necessary Scientific Basis," *IEEE Trans. Software Eng.*, Vol.20, No.3, pp.199-206, March 1994.

[6] B. Henderson-Sellers, *Object Oriented Metrics - Measures of Complexity*, Prentice Hall, 1996.

[7] S. Henry and D. Kaufra, "Software Metrics Based on Information Flow," *IEEE Trans. Software Eng.*, Vol.7, No.5, pp.510-518, Sept. 1981.

[8] D. Rombach, "Design Measurement - Some Lessons Learned," *IEEE Software*, pp.17-25, March 1990.

[9] H. Zuse, "Complexity Metrics/Analysis," *Encyclopedia of SE*, Vol.1, John Wiely & Sons, pp.131-165, 1994.

[10] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Tech. J.*, Vol.21, pp.379-423, 1948.

[11] S. Henry and C. Selig, "Predicting Source-Code Complexity at the Design Stage," *IEEE Software*, pp.36-44, March 1990.

[12] D. N. Card and W. W. Agresti, "Measuring software design complexity," *J. Systems Software*, Vol.8, pp.185-197, March 1988.

[13] W. M. Zage and D. M. Zage, "Evaluating Design Metrics on Large-Scale Software," *IEEE Software*, pp.75-80, July 1993.

[14] K. S. Lew, T. S. Dillon, and K. E. Forward, "Software Complexity and its impact on Software reliability," *IEEE Trans. Software Eng.*, Vol.14, No.11, Nov. 1988.

[15] T. J. McCabe and C. W. Butler, "Design complexity measurement and testing," *Commun. ACM*, Vol.32, pp.1415-1425, Dec. 1989.

[16] M. H. Halstead, *Elements of Software Science*, Elsevier North-Holland, 1977.

[17] W. Harrison, "An Entropy-Based Measures of Software Complexity," *IEEE Trans. Software Eng.*, Vol.18, No.12, pp. 1025-1029, Nov. 1992.

[18] R. R. Gonzalez, "A Unified Metric of Software Complexity : Measuring Productivity, Quality, and Value," *J. Systems Software*, Vol.29, pp.17-37, 1995.

[19] M. Shepperd and D. C. Ince, "A Critique of Three Metrics,"

J. Systems Software, Vol.26, pp.197-210, 1994.

[20] 홍의석, 우치수, "프로세스 복잡도에 기초한 변환 분석기의 설계 및 구현", 정보과학회논문지, 제22권 제9호, pp.1344-1353, 1995.

[21] M. P. Jones, *The practical guide to structured systems design*, Prentice Hall/Yourdon Press, 1988.

[22] B. T. Mynatt, *Software Engineering with Student Project Guidance*, Prentice Hall, 1990.

[23] E. N. Yourdon, *Modern Structured Analysis*, Prentice-Hall, 1990.

[24] A. M. Weaver, *Using the Structured Techniques A Case Study*, Yourdon Press, 1987.

[25] 홍의석, 홍성백, 김갑수, 우치수, "SDL 설계 복잡도 매트릭 집합", 정보과학회논문지(B), 제24권 제10호, pp.1053-1062, 1997.

[26] 홍의석, 정명희, "SDL 메트릭 집합의 분석적 검증", 정보처리학회논문자, 제7권 제4호, pp.1112-1121, 2000.

[27] EuySeok Hong and ChiSu Wu, "Criticality Prediction Models using SDL Metrics Set," pp.23-30, *Proc. APSEC'97/ICSC '97*, 1997.

홍 의 석

e-mail : hes@aycc.anyang.ac.kr

1992년 서울대학교 계산통계학과 졸업(학사)

1994년 서울대학교 대학원 계산통계학과 (이학석사)

1999년 서울대학교 대학원 전산과학과 (이학박사)

1999년~현재 안양대학교 디지털미디어학부 교수

관심분야 : 메트릭기반 소프트웨어 품질예측 모델, 웹기반 멀티미디어응용 기술 등

김 태 균

e-mail : ktg@taejo.pufs.ac.kr

1985년 서울대학교 지질학과 졸업(학사)

1987년 서울대학교 대학원 계산통계학과 (이학석사)

1995년 서울대학교 대학원 계산통계학과 전산과학전공(이학박사)

1988년~현재 부산외국어대학교 컴퓨터공학과 교수

관심분야 : 컴포넌트 기반 소프트웨어 공학, 객체지향 CASE 도구