객체지향 기반 소프트웨어 컴포넌트의 내부 품질 메트릭을 이용한 외부 품질 추정 기법

(Techniques to Predict External Quality from Internal Quality Metrics for Object Oriented Software Components)

박 지 환 † 신 석 규 †† 김 수 동 †††
(Ji Hwan Park) (Seok Kyoo Shin) (Soo Dong Kim)

요 약 소프트웨어 제품의 품질을 평가하기 위한 방법으로써 품질 요소, 품질 항목 및 여러 가지 메트릭을 이용한 품질 모델들이 제시되어 왔다. 하지만, 소프트웨어의 품질을 보다 정확하게 평가하기 위해서는 각각의 특징에 맞는 특화된 모델이 필요하다. 본 논문에서는 소프트웨어 컴포넌트 개발에 있어서 개발이 진행중인 상태의 소프트웨어 내부 속성에 적절한 메트릭을 적용시킨 결과를 이용하여 개발 완료 후의 소프트웨어가 가지게 되는 외부 품질을 어떻게 추정하는지에 대한 외부 품질 추정 모델을 제시한다.

소프트웨어 품질을 측정하기 위한 메트릭을 적용한 결과로써 품질 자체를 측정하는데 한정하지 않고, ISO 9126에서 제시하는 소프트웨어의 내부 속성을 이용하여 소프트웨어 컴포넌트 개발의 각 산출물에 어떻게 적용시키는지에 대한 모델을 예제를 통하여 제시한다.

키워드 : 소프트웨어 품질 모델, 품질 평가, 품질 속성, 품질 요소, 품질 항목, 품질 메트릭, 품질 추정 모델

Abstract Various quality models using quality factor, quality criteria and metrics have been proposed in order to evaluate quality of software products. However, a customized quality model which is specific to the characteristics of software component is required. In this paper, we propose external quality prediction techniques enable us to predict what external quality the final software product will have by using metrics as with internal attributes of software in development.

We also propose a model not only for measuring quality by using metrics but also for applying internal attributes of ISO 9126 into artifacts of software component development.

Key words : Software Quality Model, Quality Assessment, Quality Attribute, Quality Factor, Quality Criteria, Quality Metric, Quality Prediction Model

1. 서 론

1.1 동 기

소프트웨어 개발에 있어서 재사용성 및 범용성의 장점을 이용한 컴포넌트 기반 개발이 점차 부각되고 있다. 소프트웨어 컴포넌트를 개발하는데 있어서 컴포넌트의 품질 또한 중요하지만 이를 위한 컴포넌트의 품질 연구가 아직 미흡하다. 개발된 컴포넌트의 품질이 낮을 경우에는 소프트웨어 자체의 개발비용 보다는 컴포넌트의 개발 비용이 높은 이유로 인하여 이후에 컴포넌트를 재개발 하는데 소요되는 비용 또한 크기 때문에 고품질의 컴포넌트를 개발하는 것은 중요하다.

소프트웨어 품질을 위한 여러 가지 범용적인 품질 모델들이 제시되어 왔지만 컴포넌트를 위해 특화된 형태의 품질 모델에 대한 연구는 미흡하다. 컴포넌트를 개발함에 있어서 개발 도중의 산출물에 대한 내부 품질을 이용하여 외부 품질을 예측함으로써 개발 완료 후의 최종 품질을 예측할 수 있다. 이를 통하여 고품질의 컴포넌트를 개발할 수 있도록 유도할 수 있으며 전체 개발

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음
† 비 회 원 : 숭실대학교 컴퓨터학과
perla7410@yahoo.co.kr
†† 비 회 원 : TTA IT시험연구소 S/W시험센터
skshin@tta.or.kr
††† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@comp.ssu.ac.kr
논문접수 : 2002년 11월 5일
심사완료 : 2003년 3월 10일

비용을 감소시킬 수 있다.

본 논문에서는 컴포넌트 개발 도중의 각 산출물에 대하여 적용할 수 있는 메트릭 등을 포함한 컴포넌트 품질 모델과 내부 메트릭이 외부 메트릭과 어떠한 관계를 갖고 있는지에 대한 매핑 테이블도 제시한다. 컴포넌트 품질 모델, 내부 및 외부 메트릭 매핑 테이블을 이용한 외부 메트릭의 예측으로써 고품질 컴포넌트의 개발을 가능하게 하며 개발에 소요되는 전체 비용 또한 감소시킬 수 있다.

1.2 논문의 구성

본 논문의 2장에서는 소프트웨어 품질을 측정함에 있어서 개발 도중에 측정될 수 있는 내부 메트릭과 개발 이후에 측정 가능한 외부 메트릭, 그리고 이를 통하여 전체 품질을 예측할 수 있는 품질 추정에 대하여 살펴본다.

3장에서는 소프트웨어 컴포넌트 개발에 적용하기 위한 컴포넌트 참조 모델의 구성 요소들에 대하여 살펴보고, 4장에서는 컴포넌트 품질 평가를 위한 품질 모델을 제시하며 5장에서는 3장에서 제시한 CORBA 컴포넌트 참조 모델의 각 구성 요소에 대하여 적용될 수 있는 내부 메트릭을 제시하며 6장에서는 외부 메트릭을 제시한다.

7장에서는 3, 4, 5장을 통해서 제시된 내부 및 외부 메트릭에 대하여 각각 어떠한 요소들끼리 서로 연관 관계를 갖고 있는지에 대하여 외부 메트릭에 대한 각 영향 요소 파악 및 매핑 테이블과 품질 추정 모델을 제시한다.

마지막으로 8장에서는 이러한 내부 및 외부 메트릭 간의 매핑 테이블을 통해서 도출된 품질 추정 모델을 이용하여 CBD 산출물 각각에 대하여 내부 및 외부 메트릭이 어떻게 적용되는지 예제를 통해 나타낸다.

2. 관련 연구

2.1 소프트웨어 품질 모델[1-10]

소프트웨어의 품질을 측정하고 표현하는 모델을 소프트웨어 품질 모델이라고 하는데 대표적인 품질 모델에는 ISO 9126[11-14], McCall의 품질 모델[15], Boehm의 품질 모델[16] 및 Dromey의 품질 모델[17] 등이 있다. 이러한 품질 모델에서 사용되는 품질 측정을 위한 기본 단위 및 용어가 서로 다르다. 품질 측정을 위한 가장 상위의 단위로는 특성, 요소, 고수준 품질 특성 및 고수준 품질 속성 등이 있고 품질 요소의 하위 단위인 품질 항목으로는 부특성, 항목, 원시특성 및 품질관련 속성 등이 있다.

본 절에서는 소프트웨어 품질을 측정하기 위한 품질 모델의 대표적인 종류 네 가지에 대하여 각각의 모델에서 사용되는 품질 측정의 단위 및 분류 등에 대하여 살펴본다. 본 논문에서는 소프트웨어 품질을 측정하기 위한 기본 단위들을 McCall 모델에서 제시한 품질 요소와 품질 항목이라는 용어를 이용한다.

ISO 9126에서는 소프트웨어의 품질을 측정하기 위한 기본 단위를 크게 두 가지 즉, 6개의 품질 요소와 24개의 품질 항목으로 나누었다. ISO 9126에서의 품질 요소 및 품질 항목은 내부 품질 또는 외부 품질이라고도 하며 이 중 품질 항목은 내부 메트릭 또는 외부 메트릭으로 측정 가능하다.

McCall의 품질 모델에서는 소프트웨어의 품질 측정의 기본 단위를 11개의 품질 요소와 25개의 품질 항목으로 나누었다. ISO 9126의 품질 모델과는 다르게 가장 상위의 품질 요소들을 분류하는 기준을 제품의 운영, 수정 및 전이의 세 가지 관점에서 나누었다.

Boehm의 품질 모델에서는 품질 측정의 기본 단위를 고수준에서의 특성과 하위 수준에서의 원시 특성으로 나누었다. 상위 수준의 특성은 '일반 유틸리티', '유틸리티' 그리고 '유지보수'라는 세 가지의 관점으로부터 나뉘어 지게 된다. McCall의 품질 모델에서와 마찬가지로 특성에 대응되는 원시 특성 각각은 하나 또는 그 이상 대응 관계를 가지며 이러한 원시 특성을 측정하기 위한 메트릭도 각각 존재한다.

Dromey의 모델에서는 소프트웨어 품질을 측정하기 위한 기본 단위들을 상위의 품질 속성과 하위의 품질관련 속성으로 구분하였다. 하위 수준의 품질관련 속성을 크게 네 가지의 관점 즉, 정확성, 구조성, 모듈성 그리고 기술성 측면에서 나누었다.

본 논문에서는 앞에서 열거한 네 가지의 품질 모델에서 사용하고 있는 소프트웨어의 품질 측정을 위한 기본 단위들 중에서 McCall 모델에서 제시한 품질 요소 및 품질 항목이라는 용어를 사용하여 품질 측정을 하며, 이에 따른 각각의 세부 메트릭도 제시한다.

2.2 내부 메트릭(Internal Metric) 및 외부 메트릭 (External Metric)

내부 메트릭이란 소프트웨어 제품의 개발 도중에 측정할 수 있는 요소들에 대한 메트릭으로써, 설계 및 코딩 단계에서의 실행이 불가능한 즉, 명세서, 설계 다이어그램 또는 소스 코드 등의 산출물에 적용할 수 있는 메트릭이다.

내부 메트릭을 적용하는 목적은 개발 완료 후에 소프트웨어 제품이 갖게 되는 외부 품질 및 사용 중에 나타

날 수 있는 품질에 대한 측정을 위함이다. 이를 통하여 사용자, 평가자, 시험자 및 개발자 들은 소프트웨어 제품이 수행 가능하기 이전에 소프트웨어 제품 품질을 평가하여 개발 생명주기 프로세스 중 초기 단계에서 측정된 품질에 대한 결과로써 개발된 제품에 대한 수정 및 보완을 함으로써 제품의 질을 향상시킬 수 있도록 한다.

외부 메트릭은 실행 불가능한 개발 도중의 산출물들을 대상으로 하는 내부 메트릭과는 달리 실행 가능한 소프트웨어 또는 시스템을 측정 대상으로 한다. 개발 완료된 소프트웨어 제품을 사용하기 이전에 사용 및 비즈니스 목적에 맞는지 등을 메트릭을 통하여 평가하여야 한다. 소프트웨어 제품의 품질 요소 및 품질 항목 각각에 대하여 외부 메트릭과 연관 관계가 강한 내부 메트릭을 대상으로 소프트웨어 제품의 내부 품질 속성(Internal Quality Attribute)에 대한 정량화(Quantify)를 통하여 측정할 수 있다.

내부 메트릭을 이용하여 측정된 결과는 외부 메트릭으로 측정되는 제품 개발 완료 후에 측정 가능한 값들에 영향을 줄 수 있다. 따라서, 소프트웨어 제품의 내부 속성을 이용한 내부 메트릭을 통하여 외부 품질을 추정할 수 있는 모델 및 방법이 필요하며 측정된 각 결과가 외부 품질 요소들 중 어떠한 요소들에 직접 및 간접적으로 영향을 미치는지에 대한 대응관계 또한 제시되어야 한다. 그림 1에서의 품질을 측정하기 위한 기본 단위인 품질 요소 및 품질 항목은 가장 하위 수준에서의 메트릭으로써 측정이 가능하며, 이러한 메트릭은 개발 도중에 적용할 수 있는 내부 메트릭과 개발이 완료된 후에 적용할 수 있는 외부 메트릭으로 구성이 된다. 내부 메트릭 각각은 하나 또는 그 이상의 외부 메트릭에 영향을 미칠 수 있으므로 이러한 관계에 대한 매핑도 제시되어야 한다.

2.3 Kececi의 품질 예측 모델[18]

Kececi는 소프트웨어 제품에 대한 품질을 분석, 측정 및 평가하기 위한 논리기반의 그랙픽적 프레임워크를 제시하였다. 그랙픽적인 동적 품질 평가(Graphical Dynamic Quality Assessment, GDQA) 프레임워크는 크게 세 가지의 구성요소로 이루어져 있다. 첫째, 목적(Objectives)으로써, 사람, 하드웨어 또는 소프트웨어 속성들과 관련된 품질 요구사항으로써 측정 가능한 속성들이 인식되기까지 좀 더 세분화되어야 하고, 신뢰성(Reliability) 등이 예이다. 둘째, 기능(Functions)으로써 신뢰성 추정 모델 등이 해당이 되며, 알고리즘, 비율 또는 공식 등이 이에 포함된다. 셋째, 원시 데이타(Primary Data)로써 소프트웨어 명세서 등으로부터 얻을 수 있는 에러의 개수 등과 같은 자료들이 이에 해당된다.

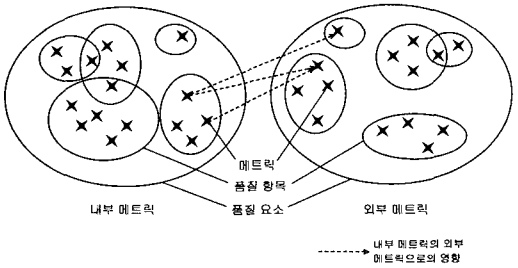

그림 1 내부 메트릭과 외부 메트릭

Kececi의 모델은 소프트웨어 품질을 측정하기 위한 품질 측정의 기본 단위를 GDQA 프레임워크를 통해서 분류하고 이에 따른 개략적인 프로세스를 소개하고 있지만, 소프트웨어 컴포넌트 등의 특화된 경우에 적용할 수 있는 특화된 모델이 필요하며 품질을 직접 측정하기 위한 메트릭 수준에서의 방법이 필요하다.

2.4 Seffah의 품질 예측 모델[19]

Seffah의 모델에서는 소프트웨어 품질 모델에서의 사용성(Usability)에 관한 메트릭을 정량화하기 위한 프레임워크인 QUIM(Quality in Use Integrated Map)을 제안하고 있는데, 크게 여섯 가지의 구성요소로 이루어져 있다. 가장 상위의 요소인 Quality in Use를 기준으로 하여 Factor, Criteria, Metric, Data 그리고 최하위의 산출물 등으로 이루어져 있다. 최하위의 구성요소인 산출물 중 Use Case, 사용자 매뉴얼 및 프로토타입 등으로부터 최상위 요소인 사용자 관점에서의 소프트웨어 품질인 Quality in Use에 이르기까지의 구성요소 및 그들간의 관계를 제시하고 있다.

다른 품질 모델에서와 마찬가지로 하위 품질 요소인 Criteria를 측정하기 위한 메트릭에 대하여 특화된 공식이 존재하지 않으며, ISO 9126에서 제시하는 범용적인 공식의 결과만을 이용하여 Criteria 및 Factor들을 측정하는데 있어서의 관계에 대한 모델만을 제시하고 있다. 따라서, 좀 더 특화된 모델 및 품질 측정을 위한 상세한 수준에서의 메트릭의 제시가 필요하다.

2.5 Dromey의 모델

Dromey의 모델에서는 소프트웨어 제품에 대한 품질을 측정하기 위한 모델에 대하여 품질 속성(Quality Attribute)과 그에 따른 하위의 요소인 Quality-Carrying Properties를 두었는데, 다른 모델에서와는 달리 하위 요소인 Quality-Carrying Properties 자체를 정확성(Correctness), 구조성(Structural), 모듈성(Modularity) 및 기술성(Descriptive) 등의 크게 네 가지 관

점으로 분류를 하였다. 또한, 하위 요소인 Quality -Carrying Properties에 대하여 영향을 미칠 수 있는 Quality Attribute를 Quality Impact라는 항목을 통하여 제시함으로써 품질 측정의 상위 요소와 하위 요소 간의 직접적인 관계를 명시하고 있다.

품질 모델의 상위 및 하위 구성요소들 간의 관계 및 영향 요소들에 대한 매핑 관계를 테이블을 통해서 잘 나타내고 있지만, 최하위 수준에서의 품질 측정 방법인 메트릭에 대한 제시가 필요하며, 모델 자체도 특정 소프트웨어 형태에 적용시킬 수 있는 특화된 형태로의 변형이 가능해야 한다.

이렇게 총 세 가지의 범용적인 소프트웨어를 위한 기존 품질 모델들에 대하여 알아보았다. 범용 품질 모델에서 제시하고 있는 품질 측정 방법 및 기본 측정 단위들은 소프트웨어의 품질을 측정하기 위한 범용적인 요소만으로 구성되어 있기 때문에 소프트웨어 컴포넌트 같은 특화된 소프트웨어에 적용하기에는 많은 어려움이 있다. 이를 위해서 범용 소프트웨어와는 달리 컴포넌트의 여러 가지 특징들이 충분히 반영된 모델이 필요하며 본 논문에서는 이러한 소프트웨어 컴포넌트에 특화하여 적용할 수 있는 품질 측정 기본 단위 및 품질 모델을 제시하고자 한다.

3. 컴포넌트 참조 모델[20-22]

본 장에서는 소프트웨어 컴포넌트에 대하여 여러 가지 내부 및 외부 메트릭을 적용하는데 있어서의 주 대상이 되는 컴포넌트를 이루는 단위 구성 요소를 알아보기 위하여 대표적인 CORBA 컴포넌트 참조 모델(CCM)[23]에 대하여 기술한다. CORBA 컴포넌트 참조 모델은 컴포넌트를 이루는 구성 요소가 CMU/SEI의 참조 모델[24]이나 EJB 모델[25] 또는 Perrone의 참조 모델[26] 보다 상세하며 다양한 장치들을 가지고 있는 포괄적인 모델이다. 또한 구현 언어에 대하여 독립적인 CORBA 자체의 특성을 포함하고 있기 때문에 구현상의 이점도 가지고 있다. 컴포넌트의 품질을 측정하기 위한 내부 및 외부 메트릭은 컴포넌트 참조 모델의 각 구성 요소에 따라 적용 되어야 하므로 명시적으로 구성 요소들을 제시하고 있는 CCM을 대표적인 컴포넌트 참조 모델로 이용한다.

그림 2에서와 같이 CORBA에 있어서의 컴포넌트(①)란 기본적인 메타 타입을 말하는 것으로써 객체 메타 타입을 확장 및 특수화한 것이다. 컴포넌트 타입은 IDL에 명세 되어 있으며 인터페이스 저장소에 나타난다. 컴포넌트는 객체 레퍼런스로 표현되는 컴포넌트 레퍼런스로 표시한다. 따라서, 컴포넌트 정의는 인터페이스에 대한 특수화 및 확장을 의미한다.

컴포넌트 타입은 IDL 컴포넌트 정의 또는 인터페이스 저장소에 의해 기술되는 특정 기능들의 집합이다. 이러한 컴포넌트 타입에 대하여 서로 다른 실행 환경(OS/하드웨어 플랫폼, 언어 등)에 여러 개의 구현이 있을 수 있지만 모두 기능은 동일해야 하며 상태 및 식별자를 갖는 개체를 생성할 수 있다.

CORBA 컴포넌트에는 Basic과 Extended 두 가지 수준의 컴포넌트가 존재한다. 두 가지 모두 컴포넌트 home에 의해 관리되지만 제공하는 것들은 서로 다르다. Basic 컴포넌트는 CORBA 객체를 'Componentize' 하기 위한 장치들을 지원하며, Extended 컴포넌트는 다양한 기능성의 집합을 제공한다.

컴포넌트를 통하여 클라이언트 혹은 어플리케이션 환경의 다른 요소들과 컴포넌트가 상호작용할 수 있도록

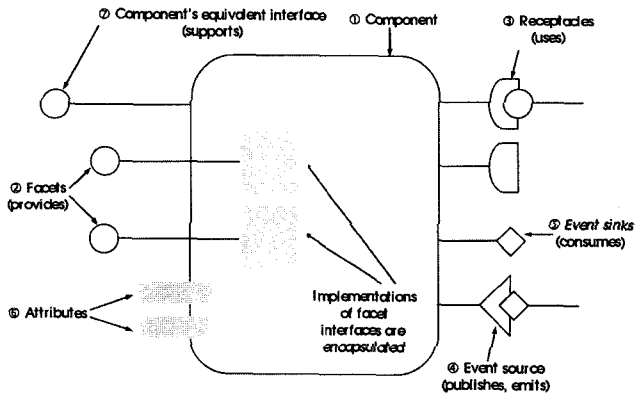

그림 2 CORBA 컴포넌트 모델

하는 표면 특성(Surface Feature)을 제공하며 포트(Port)라고 한다. CORBA 컴포넌트 모델에서는 네 가지 종류의 포트를 제공한다.

첫째, Facet(②)은 클라이언트와의 상호작용을 위하여 컴포넌트가 제공하는 인터페이스이며 둘째, Receptacle(③)은 외부의 에이전트에 의해서 제공되는 레퍼런스를 이용하기 위하여 컴포넌트의 기능을 설명하는 연결점이다. 셋째, 이벤트 소스(④)는 하나 이상의 이벤트 소비자 또는 이벤트 채널에게 이벤트를 발생시키는 연결점이며, 넷째, 이벤트 싱크(⑤)는 특정한 타입의 이벤트를 받는 연결점에 해당된다. 어트리뷰트(⑥)는 Accessor와 Mutator 오퍼레이션을 통하여 접근되는 값들이다.

컴포넌트는 Facet이라는 복수개의 객체 레퍼런스들을 제공하여 유일한 IDL 인터페이스를 지원한다. 컴포넌트는 컴포넌트의 정의를 준수한 인터페이스를 갖는 하나의 구별된 레퍼런스를 가진다. 이러한 레퍼런스가 인터페이스를 지원하며 컴포넌트의 Equivalent 인터페이스(⑦)라고 한다. 클라이언트는 이러한 Equivalent 인터페통하여 Facet에 대한 레퍼런스를 얻고 다른 컴포넌트의 Port에 연결할 수 있다. Basic 컴포넌트의 경우는 Facet을 지원하지 않기 때문에 다른 Facet으로의 항해가 불가능하며 Basic 컴포넌트의 Equivalent 인터페이스는 클라이언트와 상호작용할 수 있는 유일한 수단이 된다.

그림 2에서와 같이 Facet 인터페이스에 대한 구현은 컴포넌트에 의하여 캡슐화되어 있으며 이러한 내부 구조는 클라이언트에게 감추어져 있다. 클라이언트는 어떠한 Facet에서도 Equivalent 인터페이스로 항해할 수 있으며, Equivalent 인터페이스로부터 어떠한 Facet에 대한 레퍼런스도 얻을 수 있다. Facet의 생명주기는 Facet이 속한 해당 컴포넌트의 생명주기와 같다.

본 장에서 기술된 컴포넌트 참조 모델 중 CCM의 구성 요소들은 CMU/SEI의 참조 모델이나 EJB 모델 또는 Perrone의 참조 모델 등도 포함할 수 있기 때문에 CCM에만 국한되어 적용되지 않고 다른 컴포넌트 참조 모델에도 적용이 가능하다. 본 논문에서 사용하는 컴포넌트 참조 모델의 구성 요소에 대한 명칭은 표 1에서 정의한 약칭을 사용한다.

표 1 CCM의 구성 요소 및 약칭

	CCM의 구성 요소	약 칭
①	Component	Comp
②	Facet	IP(Provide Interface)
③	Receptacle	IR(Require Interface)
④	Event Source	EvtSrc
⑤	Event Sink	EvtSnk
⑥	Attribute	CompAttr
⑦	Component's Equivalent Interface	CompEqiIntf

4. 컴포넌트 품질 평가를 위한 품질 모델 (C-QM)

본 장에서는 3장에서 제시한 CORBA 참조 모델의 구성요소에 대하여 소프트웨어 컴포넌트 개발을 할 때 내부 메트릭 및 외부 메트릭을 적용시킬 수 있도록 품질 측정의 상위 및 하위 단위인 품질 요소 및 품질 항목 등을 포함한 품질 모델을 정의한다. 제시된 품질 모델은 ISO 9126에서 정의한 6개의 품질 요소(Characteristics) 및 22개의 품질 항목(Subcharacteristics)을 기반으로 하여 재사용성 위주의 컴포넌트를 위해 수정 및 보완하여 특화된 모델로 정의하였다.

4.1 소프트웨어 컴포넌트의 특징

본 절에서는 소프트웨어 컴포넌트가 가지는 일반적인 특징들에 대하여 관련된 품질 속성에는 어떠한 것들이 있으며, 기존의 범용 소프트웨어에 대한 품질 모델인 ISO 9126의 품질 모델과 McCall의 품질 모델에서 제시하는 관련된 품질 속성들을 살펴보며 마지막으로 본 논문에서 제시하는 C-QM에서의 관련 품질 속성에 대하여 알아본다. 표 2에서와 같이 기존 품질 모델에서 제시하는 품질 속성들의 한계점 및 C-QM과의 차이점을 통하여 소프트웨어 컴포넌트의 정확한 특징과 이를 위한 품질 속성들을 파악할 수 있다.

표 2에서와 같이 본 논문에서는 소프트웨어 컴포넌트의 특징을 크게 7가지로 분류하고 이에 대한 기존 모델에서의 품질 속성 및 C-QM에서 제시하는 품질 속성들을 비교함으로써 소프트웨어 컴포넌트가 가지는 정확한 특징 및 이를 위한 관련 속성들을 규명한다.

소프트웨어 컴포넌트의 대표적인 특징으로써 컴포넌트를 사용하게 될 다수 사용자 회사가 가지는 공통의 기능들을 컴포넌트가 수용해야만 한다. ISO 9126에서는 Functionality의 Suitability와 Compliance 품질 항목을 제시하는데, 적절한 기능 제공 측면의 Suitability와 정해진 표준에 대한 준수를 의미하는 Compliance 등을 반영하며, McCall 모델에서의 Correctness 품질 요소 중 기능에 대한 완전한 구현인 Completeness 및 수행되는 기능에 대한 범용성을 의미하는 Reusability의 Generality 등을 반영하여 기능 공통성 및 도메인 모델 준수성이라는 항목을 제시한다.

컴포넌트가 가지는 공통의 특징 이외에도 사용자가 특정 도메인에 맞도록 컴포넌트 자체를 설정할 수 있는 특화성의 경우에는 ISO 9126 및 McCall 어느 모델에서도 명시적으로 제시하고 있지 않은 항목으로써 소프트

표 2 소프트웨어 컴포넌트의 특징 및 기존 모델과의 차이점

소프트웨어 컴포넌트의 특징	관련된 품질 속성	ISO 9126의 품질 모델	McCall의 품질 모델	C QM
다수 사용자 회사의 공통 기능 제공	기능 공통성,도메인 모델준수성	Functionality의 Suitability, Compliance	Correctness의 Completeness, Reusability의 Generality	기능성의 기능 공통성, 준수성의 도메인 모델 준수성
가변성의 특화기능 제공	기능 공통성,특화성			기능성의 기능 공통성, 재사용성의 특화성
컴포넌트를 재사용의 기본 단위로 함	재사용성	Portability의 Replaceability	Maintainability의 Modularity, Self Descriptiveness	재사용성의 명세 완전성
컴포넌트 단위의 유지보수	유지보수성	Maintainability의 Changeability	Maintainability의 Simplicity	유지보수성의 변경성, 단순성
객체 지향 설계 내용에 대한 충분한 반영	기능성		Correctness의 Traceability	기능성의 적절성, 완전성
단일 컴포넌트의 기능 응집성	재사용성		Reusability의 Software System Independence	재사용성의 결합성
특정 컴포넌트 참조 모델의 표준을 준수하여 구현	준수성	Functionality의 Compliance		준수성의 컴포넌트 모델 준수성

웨어 컴포넌트의 중요한 특징 중의 하나이며, 본 논문에서는 이를 별도의 품질 항목으로써 제시한다.

소프트웨어 컴포넌트는 재사용을 목적으로 한 클래스를 그 구성 단위로 하는 특징으로 인하여 컴포넌트 자체도 재사용에 있어서의 기본 단위로써 중요한 역할을 한다. ISO 9126 모델에서는 Portability 품질 요소 중 Replaceability라는 품질 항목에서 기능적인 측면에서의 대체를 중요시하고 있으며 McCall 모델의 경우도 Maintainability의 Modularity와 Self Descriptiveness를 통하여 재사용의 단위 및 이를 위한 명세 측면을 강조하지만 컴포넌트와 같은 특정 단위에 대한 재사용을 전제로 하고 있지는 않다. 따라서, 본 논문에서는 컴포넌트를 재사용의 기본 단위로 할 때 필요한 컴포넌트 자체에 대한 명세 완전성을 별도의 품질 항목으로 제시한다.

개발이 완료된 이후의 컴포넌트에 대하여 유지보수에 있어서의 변경에 대한 용이성과 이를 위한 컴포넌트의 단순성이 중요한데 ISO 9126에서는 Maintainability의 Changeability를 통해서 변경사항에 대한 구현을 나타내며, McCall의 모델에서는 Simplicity 품질 항목에서 복잡성을 줄이는 용이한 방법을 통한 구현을 강조하고 있다.

컴포넌트의 개발시 구성요소가 되는 객체지향 모델링의 결과들이 구현된 컴포넌트를 통하여 얼마나 반영되는지에 대한 정도를 위하여 본 논문에서는 기능성 품질 요소 중 적절성과 완전성을 제시하며 McCall 모델에서는 Correctness의 Traceability를 통하여 요구사항에서부터 구현에 이르는 일련의 추적성을 강조하고 있다.

McCall 모델에서 제시하는 Software System Independence는 Portability 및 Reusability 측면에 적용될 수 있는 품질 항목으로써 소프트웨어 제품의 독립성을 강조한다. 본 논문에서는 이를 위한 품질 항목으로써 재사용성의 결합성을 제시한다.

마지막으로 소프트웨어 컴포넌트는 특정 컴포넌트 참조 모델의 구성 요소들을 기반으로 하여 구현되어야 하는데, McCall 모델에서는 이를 위한 항목을 명시적으로 제시하고 있지 않으며 ISO 9126 모델에서는 범용적 준수성인 Compliance를 통하여 나타내고 있다.

위와 같은 총 7가지의 소프트웨어 컴포넌트의 특징들을 기반으로 하여 4.2와 4.3에서는 C-QM의 품질 요소 및 품질 항목들을 제시한다.

4.2 C-QM의 품질 요소

기능성(Functionality) : 소프트웨어 제품의 품질에 있어서의 기능성은 요구사항을 통해 명시되거나 또는 암시되어 있는 기능들을 제공하는 것으로써 이러한 요구사항의 기능들이 충분히 구현이 되었는지 또는 구현된 정도가 얼마나 정확한지 등을 측정함으로써 기능성이라는 품질 요소를 판단한다[1]. 소프트웨어 컴포넌트 개발에 있어서는 컴포넌트의 특성상 여러 가지 요구사항의 공통된 집합인 패밀리 요구사항에 명시되어 있거나 또는 암시된 기능들을 만족하는 속성이다. 본 논문에서는 기능성 품질 요소를 측정하기 위하여 적절성, 완전성 및 기능 공통성의 세 가지 품질 항목을 이용한다.

재사용성(Reusability) : 재사용성 품질 요소는 소프트웨어 품질 측정을 위한 범용 품질 모델을 제시하고 있는 ISO 9126에서는 품질 요소 및 품질 항목의 어떠한 형태로도 명시적으로는 제시하고 있지 않은 부분이지만, 소프트웨어 컴포넌트가 재사용성을 큰 특징으로 가지기 때문에 가장 중요한 요소 중의 하나이다. 이러한 재사용성 품질 요소를 측정하기 위한 세부 품질 항목으로써 특화성, 응집성, 결합성 및 명세 완전성을 제시한다.

유지보수성(Maintainability) : 소프트웨어 컴포넌트는 클래스 다이어그램, Use Case 다이어그램 및 시퀀스 다이어그램과 같은 객체 지향 개발 산출물들을 이용하여 단위 구성 요소들이 클래스라는 점과 컴포넌트 단위로 재사용 할 수 있다는 점에서 유지 보수성이 중요시된다. 클래스 자체도 유지 보수를 위한 어트리뷰트와 오퍼레이션의 분리를 통해서 이루어지므로 이러한 클래스를 구성 단위로 하는 컴포넌트 역시 유지보수적인 측면이 중요시되어야 한다. 이러한 소프트웨어 컴포넌트의 유지보수성을 측정하기 위한 품질 항목으로 변경성 및 단순성을 이용한다.

소프트웨어 컴포넌트는 개발 이후에 범용적으로 여러 도메인에서 공통적으로 사용되며 또한 기능상의 수정 또는 보완 등과 같은 변경 시에도 변경 내용이 컴포넌트에 적절하게 반영되어야 한다. 이러한 점은 컴포넌트의 기능에 대한 변경을 얼마나 용이하게 할 수 있는 지와 컴포넌트가 변경에 용이하도록 어느 정도 단순하게 구성되어 있는지를 측정함으로써 판단할 수 있다.

준수성(Conformance) : 준수성은 소프트웨어 컴포넌트를 개발함에 있어서 지켜져야 하는 일종의 규약이나 표준을 준수함으로써 이후의 유지 보수에도 용이할 수 있도록 하기 위한 품질 요소로써 주어진 도메인에서 비즈니스 측면에서의 규약 등을 지켰는가 하는 도메인 모델 준수성과 개발된 소프트웨어 컴포넌트가 컴포넌트 참조 모델을 준수하여 개발되었는지 등을 측정하는 컴포넌트 모델 준수성으로써 판단된다.

4.3 C-QM의 품질 항목

본 절에서는 C-QM에서 정의한 네 가지의 품질 요소인 기능성, 재사용성, 유지보수성 및 준수성 각각에 대하여 하위 수준에서 총 10개의 품질 항목을 제안한다.

기능성의 경우에는 패밀리 요구사항에서 식별된 기능들이 설계 다이어그램으로 충분히 반영되었는지를 위한 적절성, 설계 다이어그램들 간의 일관성이 잘 지켜졌는지에 대한 완전성 및 각 요구사항들의 기능들이 공통기능으로 추출되어 패밀리 요구사항에 반영되었는지에 대한 기능 공통성을 기능성의 품질 항목으로 제시한다.

소프트웨어 컴포넌트의 재사용성을 위해서는 공통된 기능 이외의 가변성에 대한 설정을 할 수 있는 특화성, 컴포넌트들 간의 상호작용에 대한 결합성 및 재사용에 있어서의 용이함을 위한 명세 완전성 등이 제시된다. 유지보수 측면에서는 개발완료 이후의 수정 및 변경 사항에 대하여 충분히 반영될 수 있는지에 대한 변경성 및 변경을 위한 구조상의 단순성을 제시한다.

준수성에서는 주어진 표준을 얼마나 구현된 컴포넌트가 준수하는가에 대한 것으로써 동일한 도메인들을 대상으로 한 도메인 모델 준수성과 구현을 위한 표준인 컴포넌트 모델 준수성 두 가지를 제시한다.

적절성(Suitability) : 적절성은 명시된 패밀리 요구사항의 기능이 적절하게 제공되는지 또는 전체 요구사항

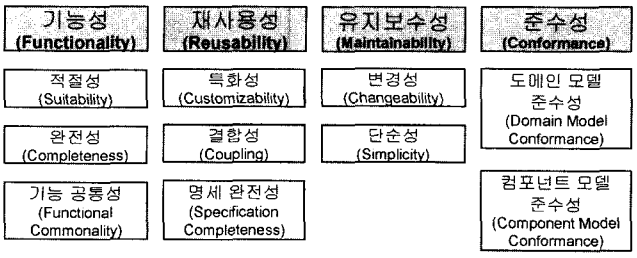

그림 3 컴포넌트 품질 평가를 위한 품질 요소 및 품질 항목

중에 구현된 기능들의 비율이 어느 정도인지를 측정하게 된다. 소프트웨어 컴포넌트를 개발하는 과정에서의 적절성은 패밀리 요구사항에 명시 또는 암시되어 있는 기능들이 개발 과정중의 초기 산출물인 설계 다이어그램을 통해서 반영된 정도와 구현된 컴포넌트에 이러한 기능들이 반영된 정도를 측정한다.

완전성(Completeness) : 완전성은 패밀리 요구사항에 대하여 구현된 기능이 완전한가를 판단하기 위한 항목으로써 개발 초기 패밀리 요구사항의 기능들이 설계 다이어그램 상에 반영된 산출물들을 대상으로 하여 산출물들 사이에 일관성 있는 기능성의 반영이 이루어져 있는가를 측정한다.

기능 공통성(Functional Commonality) : 기능 공통성은 동일한 도메인을 대상으로 하는 여러 가지 요구사항의 집합 중에서 개발하려는 소프트웨어 컴포넌트에 공통적인 기능들이 추출되어 구현되었는지를 측정하기 위한 것으로써 재사용 위주의 컴포넌트 특징 측면에서 볼 때, 공통적으로 사용되는 기능들을 수용함으로써 기능성 및 재사용성을 높일 수 있다.

특화성(Customizability) : 특화성은 소프트웨어 컴포넌트의 가변성 부분을 이용하여 특정 도메인에 맞게 재구성할 수 있는 부분에 대한 품질 특성으로써 공통 기능 이외에 특화 할 수 있는 부분을 측정하는 품질 항목이다.

결합성(Coupling) : 결합성은 하나의 컴포넌트를 대상으로 하여 제공되는 기능이 얼마나 적절하게 구성되어 있는가에 대한 응집성과는 다르게 컴포넌트 각각이 다른 컴포넌트와의 연동 시에 컴포넌트 간에 발생하는 상호작용이 적음으로 인하여 해당 컴포넌트 자체의 기능이 독립적으로 수행될 수 있음을 나타내는 항목이다.

명세 완전성(Specification Completeness) : 개발된 소프트웨어 컴포넌트가 다른 시스템 또는 어플리케이션의 개발에 이용될 경우 컴포넌트의 재사용 측면에서의 용이성을 위하여 컴포넌트 자체에 대한 명세가 얼마나 완전한지를 측정한다. 명세가 완전할수록 명세를 통한 개발된 컴포넌트의 이해가 용이하여 재사용에 있어서의 비용을 절감할 수 있다.

변경성(Changeability) : 변경성은 단위 컴포넌트에 대하여 유지 보수자의 관점에서 어떠한 기능상의 수정 또는 보완과 같은 변경사항이 있을 경우 해당 변경 내용이 적절하게 반영되었는지에 대한 여부를 측정한다.

단순성(Simplicity) : 소프트웨어 컴포넌트에 대하여 기능 등의 변경 내용 자체가 소스 코드 및 명세서 또는 개발 완료된 제품에도 용이하게 이루어져야 한다. 단순성은 이러한 변경을 함에 있어서 구조 또는 기능상의 단순함으로 인하여 유지보수가 용이한가에 대한 판단의 기준이 된다.

도메인 모델 준수성(Domain Model Conformance) : 도메인 모델 준수성은 개발하려는 대상 소프트웨어 컴포넌트의 도메인 영역 즉, 은행, 병원 등과 같은 비즈니스적인 범위 내에서 공통적이며 필수적으로 준수해야 하는 사항들을 지키는지에 대한 특성이다.

컴포넌트 모델 준수성(Component Model Conformance) : 컴포넌트 모델 준수성은 정해진 컴포넌트 참조 모델에 대하여 각 구성 요소 및 규약 등 개발된 소프트웨어 컴포넌트가 충분히 준수하여 개발되었는지에 대한 특성이다.

5. C-QM의 내부 메트릭

본 장에서는 3장에서 제시한 컴포넌트 참조 모델을 이루는 여러 가지 구성 요소들을 대상으로 소프트웨어 컴포넌트를 개발하는데 있어서 개발 도중에 측정될 수 있는 내부 메트릭을 정의한다.

5.1 적절성(Suitability)

적절성은 주어진 요구사항의 공통된 기능들을 추출한 패밀리 요구사항에서 암시 또는 명시된 기능들이 설계 내용으로 반영이 되었는지에 대한 정도를 측정한다. 그림 4에서와 같이 OOA/D의 초기 산출물인 요구사항 명세서로부터 Use Case 다이어그램 및 클래스 다이어그램으로의 설계 내용을 통하여 측정할 수 있다. 소프트웨어 컴포넌트 개발을 위한 가장 선행되는 과정이며 공통된 기능들의 집합인 패밀리 요구사항의 기능들이 분석 모델을 통하여 나타나는 첫 번째 과정에서의 산출물들이므로 이들을 측정하는 것은 중요하다.

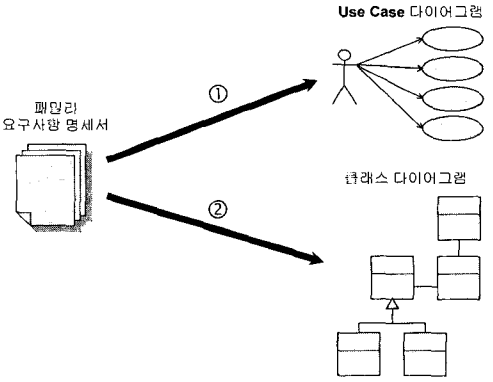

그림 4 적절성 측정을 위한 OOA/D 초기 산출물들 간의 관계

그림 4에서와 같이 OOA/D의 초기 산출물인 패밀리 요구사항 명세서의 기능들이 각각 Use Case 다이어그램 및 클래스 다이어그램의 기능들로 반영되며 적절성 측정을 위한 메트릭에 입력물로써 사용된다. 표 3에서는 적절성을 측정하기 위해 필요한 메트릭과 해당 메트릭이 적용될 산출물들을 나타낸다.

표 3 적절성 측정을 위한 입력물

	메트릭	입력물 A	입력물 B
①	$Suitability_{FamilyReqSpecToUC}(STRU)$	패밀리 요구사항 명세서	Use Case 다이어그램
②	$Suitability_{FamilyReqSpecToCD}(STRC)$	패밀리 요구사항 명세서	클래스 다이어그램

$Suitability_{FamilyReqSpecToUC}(STRU)$: 패밀리 요구사항으로부터 식별된 기능과 Use Case 다이어그램 상에 나타난 Use Case와의 관계를 측정하기 위한 메트릭.

$$Suitability_{FamilyReqSpecToUC}(STRU) = 1 - \frac{\sum_{i=1}^{n} Fn_{NotAppearedAsInUC}{}^{i}}{\sum_{i=1}^{n} Fn_{IdentifiedInFamilyReqSpec}{}^{i}}$$

수식 1 패밀리 요구사항과 Use Case 다이어그램 간의 적절성을 위한 내부 메트릭

$\sum_{i=1}^{n} Fn_{NotAppearedAsInUC}{}^{i}$ 은 패밀리 요구사항에서 하나의 기능으로서 식별되었으나 Use Case 다이어그램 상에는 나타나지 않은 기능들의 개수이며,

$\sum_{i=1}^{n} Fn_{IdentifiedInFamilyReqSpec}{}^{i}$ 은 패밀리 요구사항에서 식별된 기능들의 개수로서 직접 명시되어 있거나 또는 암시되어 있는 기능들을 모두 포함한다.

패밀리 요구사항에서 식별된 기능들과 Use Case 다이어그램 상에 나타나는 Use Case는 Granularity에 있어서 차이를 가질 수 있으며, 두 가지 경우에 있어서의 기능의 개수를 동일한 Granularity에서 Count해야 한다. 결과값이 1에 가까울수록 패밀리 요구사항에서 식별된 기능들이 Use Case 다이어그램 상에서도 하나의 Use Case로서 그려져서 패밀리 요구사항으로부터 Use Case 다이어그램으로의 매핑이 올바르게 이루어졌음을 의미한다.

$Suitability_{FamilyReqSpecToCD}(STRC)$: 패밀리 요구사항에서 식별된 기능들이 클래스 다이어그램 상의 각 클래스가 가지는 오퍼레이션을 통해 충분히 반영되었는지를 측정하기 위한 메트릭.

$$Suitability_{FamilyReqSpecToCD}(STRC) = 1 - \frac{\sum_{i=1}^{n} Fn_{NotAppearedAsClassOp}{}^{i}}{\sum_{i=1}^{n} Fn_{IdentifiedInFamilyReqSpec}{}^{i}}$$

수식 2 패밀리 요구사항과 클래스 오퍼레이션간의 적절성을 위한 내부 메트릭

$\sum_{i=1}^{n} Fn_{NotAppearedAsClassOp}{}^{i}$ 은 패밀리 요구사항에서 하나의 기능으로 식별하였으나 클래스 다이어그램 상의 클래스의 오퍼레이션을 통하여 반영되지 않은 오퍼레이션들의 개수이며, $\sum_{i=1}^{n} Fn_{IdentifiedInFamilyReqSpec}{}^{i}$ 은 패밀리 요구사항으로부터 식별된 기능들의 개수로서 직접 명시되어 있거나 또는 암시되어 있는 기능들을 모두 포함한다.

패밀리 요구사항에서 식별된 기능들과 클래스 다이어그램 상에 나타나는 클래스의 오퍼레이션도 Granularity 상의 차이를 가질 수 있으며, 두 가지 경우에 있어서의 기능의 개수를 적절한 Granularity에서 Count해야 한다. 결과값이 1에 가까울수록 요구사항에서 식별된 기능들이 클래스 다이어그램 상의 클래스 오퍼레이션으로의 매핑이 충분히 이루어졌음을 의미한다.

5.2 완전성(Completeness)

완전성은 패밀리 요구사항으로부터 작성된 Use Case 다이어그램, 클래스 다이어그램 및 시퀀스 다이어그램들 간의 관계를 통해서 설계 산출물들에 반영된 기능들간에 일관성이 유지되었는지를 확인한다. 이를 위해서 그림 5에서와 같이 각각의 다이어그램 상에 나타나는 기능, 오퍼레이션 또는 메시지 등을 비교함으로써 기능에 대한 설계 내용이 완전한지를 확인한다.

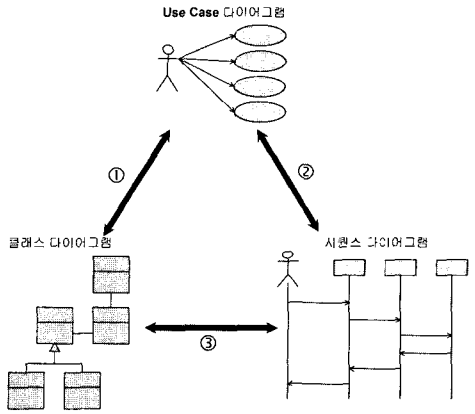

그림 5 완전성 측정을 위한 OOA/D 초기 산출물들 간의 관계

그림 5에서와 같이 OOA/D의 산출물인 Use Case 다이어그램, 클래스 다이어그램 및 시퀀스 다이어그램은 완전성 측정을 위한 메트릭에 입력물로써 이용된다. 표 4에서는 완전성을 측정하기 위해 필요한 메트릭과 해당 메트릭이 적용될 OOA/D의 각 산출물들을 나타낸다.

표 4 완전성 측정을 위한 입력물

	메트릭	입력물 A	입력물 B
①	$Completeness_{UCToCD}(CTUC)$ $Completeness_{CDToUC}(CTCU)$	Use Case 다이어그램	클래스 다이어그램
②	$Completeness_{UCToSeq}(CTUS)$ $Completeness_{SeqToUC}(CTSU)$	Use Case 다이어그램	시퀀스 다이어그램
③	$Completeness_{SeqToCD}(CTSC)$ $Completeness_{CDToSeq}(CTCS)$	클래스 다이어그램	시퀀스 다이어그램

$Completeness_{UCToCD}(CTUC)$: Use Case 다이어그램 상에 나타난 기능들인 Use Case와 클래스 다이어그램 상에 나타난 클래스 오퍼레이션 간의 관계를 통해서 일관된 설계가 이루어졌는지를 확인할 수 있는 메트릭.

$Completeness_{CDToUC}(CTCU)$: 클래스 다이어그램 상의 오퍼레이션들이 나타내는 기능성들이 Use Case 다이어그램 상의 기능들로 잘 반영되었는지를 측정하며, 클래스 다이어그램 상의 클래스 오퍼레이션이 기준이 됨.

$$Completeness_{UCToCD}(CTUC) = 1 - \frac{\sum_{i=1}^{n} Fn_{NotAppearedAsClassOpInCD}{}^{i}}{\sum_{i=1}^{n} Fn_{IdentifiedAsClassOpInUC}{}^{i}}$$

수식 3 Use Case 다이어그램과 클래스 다이어그램간의 완전성을 위한 내부 메트릭

$\sum_{i=1}^{n} Fn_{NotAppearedAsClassOpInCD}{}^{i}$은 Use Case 다이어그램을 통하여 클래스의 오퍼레이션으로 식별된 기능들 중 클래스 오퍼레이션으로 나타나지 않은 기능들의 개수이며, $\sum_{i=1}^{n} Fn_{IdentifiedAsClassOpInUC}{}^{i}$은 Use Case 다이어그램의 Use Case들 중 클래스 다이어그램의 클래스 오퍼레이션으로 식별되는 기능들의 개수이다.

Use Case 다이어그램 상의 기능들의 개수와 클래스 다이어그램 상의 클래스 오퍼레이션의 개수에는 차이가 있을 수 있기 때문에 두 가지 경우에서의 기능과 오퍼레이션의 개수를 동일한 수준에서 Count 해야 한다. 결과값이 1에 가까울수록 Use Case 다이어그램 상에서 Use Case를 통하여 식별할 수 있는 기능들이 클래스 다이어그램의 클래스 오퍼레이션으로의 매핑이 올바르게 이루어졌음을 의미한다.

$Completeness_{UCToSeq}(CTUS)$은 Use Case 다이어그램 상에서 주요한 기능을 수행하는 각 Use Case들이 시퀀스 다이어그램으로 작성되었는지를 판단할 수 있는 메트릭이며, $Completeness_{SeqToUC}(CTSU)$은 시퀀스 다이어그램으로 나타난 하나의 기능에 대하여 Use Case 다이어그램 상의 Use Case로 존재하는지를 측정하며, 시퀀스 다이어그램이 기준이 된다.

$$Completeness_{UCToSeq}(CTUS) = 1 - \frac{\sum_{i=1}^{n} Fn_{NotDescribedAsInSeq}{}^{i}}{\sum_{i=1}^{n} Fn_{IdentifiedAsSignificantInUC}{}^{i}}$$

수식 4 Use Case 다이어그램과 시퀀스 다이어그램 간의 완전성을 위한 내부 메트릭

$\sum_{i=1}^{n} Fn_{NotDescribedAsInSeq}{}^{i}$은 Use Case 다이어그램에서는 전체 시스템에 있어서의 주요한 기능들로 식별된 기능들 중에서 시퀀스 다이어그램을 통하여 나타나지 않은 기능들의 개수이며, $\sum_{i=1}^{n} Fn_{IdentifiedAsSignificantInUC}{}^{i}$은 Use Case 다이어그램에 나타난 Use Case들 중에서 전체 시스템에 있어서 주요한 핵심 기능에 해당되는 것으로써 시퀀스 다이어그램을 통하여 기능 수행에 필요한 메시지 흐름 등이 다이어그램으로 나타날 필요가 있는 기능들의 개수이다. 결과값이 1에 가까울수록 Use Case 다이어그램에서 핵심 기능으로 식별된 기능들이 시퀀스 다이어그램으로 작성되었음을 의미한다.

$Completeness_{SeqToCD}(CTSC)$: 시퀀스 다이어그램 상의 객체들 사이에서 주고 받는 메시지들이 클래스 다이어그램 상의 해당 클래스의 오퍼레이션으로 존재 하는지를 확인함으로써 시퀀스 다이어그램과 클래스 다이어그램 상의 일관성을 확인하는 메트릭.

$Completeness_{CDToSeq}(CTCS)$: 클래스 다이어그램상에 존재하는 클래스 오퍼레이션이 시퀀스 다이어그램 상의 메시지로 존재하는지를 측정하는지를 측정하며 클래스 다이어그램이 기준이 됨.

$$Completeness_{SeqToCD}(CTSC) = 1 - \frac{\sum_{i=1}^{n} Msg_{NotExistAsClassOpInCD}{}^{i}}{\sum_{i=1}^{n} Msg_{InSeq}{}^{i}}$$

수식 5 시퀀스 다이어그램과 클래스 다이어그램 간의 완전성을 위한 내부 메트릭

$\sum_{i=1}^{n} Msg_{NotExistAsClassOpInCD}{}^{i}$ 은 시퀀스 다이어그램에 나타난 메시지들 중 클래스 다이어그램 상의 클래스 오퍼레이션으로 존재하지 않는 메시지들의 개수이며, $\sum_{i=1}^{n} Msg_{InSeq}{}^{i}$ 은 시퀀스 다이어그램에 나타난 메시지들의 개수이다. 결과값이 1에 가까울수록 시퀀스 다이어그램 상에 나타난 메시지들이 클래스 다이어그램 상의 클래스 오퍼레이션으로도 존재함을 의미한다.

5.3 기능 공통성(Functional Commonality)

기능 공통성은 그림 6에서와 같이 주어진 도메인 영역 내에서 여러 가지 요구사항들의 집합을 대상으로 하여 공통된 요구사항인 패밀리 요구사항에 대하여 해당 컴포넌트를 개발하게 되는데, 각 요구사항들이 가지고 있는 공통적인 기능들이 개발하려는 컴포넌트에 얼마나 잘 반영되었는지에 대한 정도이다.

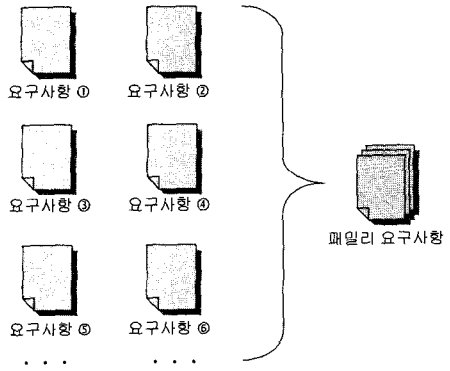

그림 6 패밀리 요구사항

각각의 요구사항에서 제시된 기능들과 개발된 컴포넌트가 가지게 될 기능간의 관계를 요구 사항의 기능들을 비교함으로써 그 정도를 파악할 수 있다.

$$FunctionalCommonalityInt(FCINT) = 1 - \frac{\frac{1}{n}\left(\sum_{i=1}^{n} Fn_{NotIncludedInFamilyReqSpec}{}^{i}\right)}{\frac{1}{n}\left(\sum_{i=1}^{n} Fn_{ReqSpec}{}^{i}\right)}$$

수식 6 기능 공통성을 위한 내부 메트릭

$\frac{1}{n}\left(\sum_{i=1}^{n} Fn_{NotIncludedInFamilyReqSpec}{}^{i}\right)$ 은 각 요구사항들의 기능 중 공통된 기능들이 추출되어 반영된 패밀리 요구사항에 포함되지 않은 기능들의 평균 개수이며, $\frac{1}{n}\left(\sum_{i=1}^{n} Fn_{ReqSpec}{}^{i}\right)$ 은 전체 요구사항들의 집합을 대상으로 하여 각 요구사항이 가지는 평균 기능들의 개수이다. 결과값이 1에 가까울수록 공통 요구사항이 각각의 요구사항에 포함되어 있는 기능들을 많이 포함하고 있음을 의미한다.

5.4 특화성(Customizability)

특화성은 컴포넌트가 가지는 특성 중의 하나인 가변성을 이용하여 컴포넌트를 사용목적 또는 특정 도메인에 맞도록 가변성 부분을 설정 후 사용할 수 있는 특성으로써 본 논문에서는 Use Case 다이어그램 상에 나타나 있는 전체 Use Case의 개수와 Use Case 다이어그램 상에서 가변성을 포함한 Use Case의 개수 비교를 통하여 판단할 수 있다.

$$CustomizabilityInt(CZINT) = 1 - \frac{\sum_{i=1}^{n} Fn_{VarNotProvidedAsInUC}{}^{i}}{\sum_{i=1}^{n} Fn_{TotalAsInUC}{}^{i}}$$

수식 7 특화성 측정을 위한 내부 메트릭

$\sum_{i=1}^{n} Fn_{VarNotProvidedAsInUC}{}^{i}$ 은 Use Case 다이어그램 상의 기능들 중에서 가변성을 포함하고 있지 않은 Use Case의 개수이며, $\sum_{i=1}^{n} Fn_{TotalAsInUC}{}^{i}$ 은 Use Case 다이어그램 상에 나타난 전체 Use Case의 개수이다. 특화를 함으로써 고유의 기능을 많이 가지게 된다면 상대적으로 범용성은 떨어지지만 특정 도메인의 목적에 맞는 기능들을 가지게 된다.

5.5 결합성(Coupling)

결합성은 소프트웨어 컴포넌트에 있어서 컴포넌트를 구성하는 단위인 클래스들 간의 연관 관계가 서로 다른 컴포넌트 사이에서 얼마나 존재하는가를 측정함으로써 판단할 수 있다. 전체 클래스 다이어그램 상의 클래스들 가운데 Generalization, Aggregation, Composition, Dependency, Association 등과 같이 서로 다른 클래스와 직접적으로 관계를 맺고 있는 클래스들이 하나의 단일 컴포넌트 내에 존재할 수 있는지를 측정한다. 개발 중간 산출물인 컴포넌트 다이어그램 상의 독립된 컴포넌트 내에 존재하는 구성 클래스들간의 관계로써 파악할 수 있다.

$$CouplingInt(CPINT) = \frac{\sum_{i=1}^{n} ClassRel_{AsWithAnotherCompDgm}{}^{i}}{\sum_{i=1}^{n} ClassRel_{InCompDgm}{}^{i}}$$

수식 8 결합성 측정을 위한 내부 메트릭

$\sum_{i=1}^{n} ClassRel_{AsWithAnotherCompDgm}{}^{i}$ 은 컴포넌트 다이어그램 상

에서 하나의 컴포넌트 내에 존재하는 클래스가 가지는 클래스들 간의 연관 관계가, 중복된 클래스의 할당으로 인하여 다른 컴포넌트와의 관계에서도 중복되어 나타나는 즉, 하나의 클래스가 복수 개의 컴포넌트 상에 존재하거나 또는 하나의 컴포넌트가 다른 컴포넌트의 기능들을 이용 함으로써 컴포넌트 간의 상호작용이 많은 경우에 있어서 연관 관계의 개수이며, $\sum_{i=1}^{n} ClassRel_{InCompDgm}{}^{i}$ 은 컴포넌트 다이어그램 상에서 하나의 컴포넌트 내에 존재하는 클래스들 사이의 연관 관계의 개수로써 Generalization, Aggregation, Composition, Dependency 및 Association 등을 모두 포함한다.

결과값이 클수록 구성 클래스들이 단일 컴포넌트 내에만 존재하여 컴포넌트 자체가 결합성이 낮음을 의미한다.

5.6 명세 완전성(Specification Completeness)

명세 완전성은 컴포넌트의 개발 중간 산출물에 해당되는 Use Case 다이어그램, 클래스 다이어그램 및 시퀀스 다이어그램 등에 대하여 명세서를 통한 명세가 충분히 이루어졌는지를 측정한다. 설계 내용 즉, 각 다이어그램 상의 내용들이 명세서를 통하여 기술됨으로써 이후에 설계 내용에 대한 재사용이 용이하다.

$$SpecificationCompletenessInt(SCINT) = 1 - \frac{\left(\sum_{i=1}^{n} UseCase_{NotAsInUCDesc}{}^{i}\right) + \left(\sum_{i=1}^{n} Class_{NotAsInSpec}{}^{i}\right) + \left(\sum_{i=1}^{n} Seq_{NotAsInSpec}{}^{i}\right)}{\left(\sum_{i=1}^{n} UseCase_{AsInUC}{}^{i}\right) + \left(\sum_{i=1}^{n} Class_{AsInCD}{}^{i}\right) + \left(\sum_{i=1}^{n} Seq_{AsInSeq}{}^{i}\right)}$$

수식 9 명세 완전성을 위한 내부 메트릭

$\sum_{i=1}^{n} UseCase_{NotAsInUCDesc}{}^{i}$ 은 Use Case 다이어그램 상에 나타난 Use Case들 중 Use Case Description을 통하여 나타나지 않은 Use Case의 개수이며, $\sum_{i=1}^{n} Class_{NotAsInSpec}{}^{i}$ 은 클래스 다이어그램 상의 클래스들 중 명세서에 명세 되지 않은 클래스의 개수이고, $\sum_{i=1}^{n} Seq_{NotAsInSpec}{}^{i}$ 은 시퀀스 다이어그램으로 나타난 시퀀스 중 명세서에 명세되지 않은 시퀀스의 개수이며, $\sum_{i=1}^{n} UseCase_{AsInUC}{}^{i}$ 은 Use Case 다이어그램 상에 나타난 Use Case의 개수이다. $\sum_{i=1}^{n} Class_{AsInCD}{}^{i}$ 은 클래스 다이어그램 상에 나타난 클래스의 개수이며, $\sum_{i=1}^{n} Seq_{AsInSeq}{}^{i}$ 은 시퀀스 다이어그램의 개수이다. 결과값이 1에 가까울수록 설계 내용이 명세서를 통하여 명세가 충분히 되었음을 의미한다.

5.7 변경성(Changeability)

변경성은 구현된 기능에 대하여 이후에 수정 또는 보완과 같은 변경이 있을 경우에 변경된 내용이 적용된 이후에도 변경되기 이전과 동일한 성능 효과를 유지 또는 향상시킬 수 있도록 하는 것이며 변경된 내용이 코드 상의 에러 등을 일으키지 않아야 한다.

$$ChangeabilityInt(CBINT) = 1 - \frac{\sum_{i=1}^{n} Fn_{UnsuccessfullyChangedInSourceCode}{}^{i}}{\sum_{i=1}^{n} Fn_{RequiredToChangeInFamilyReqSpec}{}^{i}}$$

수식 10 변경성 측정을 위한 내부 메트릭

$\sum_{i=1}^{n} Fn_{UnsuccessfullyChangedInSourceCode}{}^{i}$ 은 패밀리 요구사항 명세서를 통하여 변경을 한 후 의도된 대로 소스 코드 상으로 반영되지 않은 기능들의 개수를 측정하며, $\sum_{i=1}^{n} Fn_{RequiredToChangeInFamilyReqSpec}{}^{i}$ 은 구현된 컴포넌트의 기능성에 대하여 변경이 요구되는 기능의 개수로서 대상은 패밀리 요구사항 명세서 상의 기능 단위로 한다. 결과값이 1에 가까울수록 요구사항 명세서 상에서 변경한 내용들이 의도한 대로 반영되었음을 의미한다.

5.8 단순성(Simplicity)

단순성은 유지보수적인 측면에서 컴포넌트의 기능단위 별로 수정 또는 보완하고자 하는 내용이 있을 경우에 소스 코드 상에서 얼마나 많은 부분을 변경해야 하는지를 측정하는 품질 항목이다.

$$SimplicityInt(SPINT) = \frac{\sum_{i=1}^{n} Fn_{ModifiedInSourceCode}{}^{i}}{\sum_{i=1}^{n} Fn_{ModifiedInFamilyReqSpec}{}^{i}}$$

수식 11 단순성 측정을 위한 내부 메트릭

$\sum_{i=1}^{n} Fn_{ModifiedInSourceCode}{}^{i}$ 은 변경이 필요한 기능들에 대하여 수정이 일어난 소스 코드 상의 함수의 개수이며, $\sum_{i=1}^{n} Fn_{ModifiedInFamilyReqSpec}{}^{i}$ 은 패밀리 요구사항 명세서를 통하여 기능상 변경을 한 기능들의 개수이다. 결과값이 1일 경우는 하나의 기능에 대하여 소스 코드 상에서 하나의 기능에 대하여 하나의 함수만 수정하면 되므로 유지보수가 용이함을 의미한다.

5.9 도메인 모델 준수성(Domain Model Conformance)

도메인 모델 준수성은 여러 가지 요구사항들의 집합

에서 공통적인 기능성들을 위주로 하여 재구성한 패밀리 요구사항에 대하여 도메인 모델에서 제시하는 주요한 비즈니스적인 기능들을 충분히 준수하며 포함하는지를 측정한다.

$$DMConformanceInt(DCINT) = 1 - \frac{\sum_{i=1}^{n} Fn_{DMRequiredNotAsInUC}{}^{i}}{\sum_{i=1}^{n} Fn_{RequiredInDM}{}^{i}}$$

수식 12 도메인 모델 준수성 측정을 위한 내부 메트릭

$\sum_{i=1}^{n} Fn_{DMRequiredNotAsInUC}{}^{i}$: 도메인 모델에서 필수적으로 요구되는 기능들 중에서 Use Case 다이어그램 상에 독립된 Use Case들로 나타나지 않은 기능들의 개수.

$\sum_{i=1}^{n} Fn_{RequiredInDM}{}^{i}$: 도메인 모델에서 필수적으로 요구되는 기능들 즉, 동일한 도메인에서는 반드시 제공되어야 하는 주요한 기능들의 개수. 결과값이 1에 가까울수록 해당 도메인에서 필수적으로 요구되는 기능들이 Use Case 다이어그램 상에서 하나의 기능에 해당되는 Use Case로 반영이 되었음을 의미한다.

5.10 컴포넌트 모델 준수성(Component Model Conformance)

소프트웨어 컴포넌트 개발에 있어서 기준이 되는 컴포넌트 참조 모델이 필요하며 이러한 컴포넌트 참조 모델의 각 구성 요소들을 정확히 준수하여 개발하였는지를 측정하는 것은 중요하다. 컴포넌트 모델 준수성이라는 품질 항목을 통하여 개발된 컴포넌트가 표준으로 사용한 컴포넌트 참조 모델의 구성 요소들을 얼마나 준수하였는지를 측정한다.

$$CMConformanceInt(CCINT) = 1 - \frac{\sum_{i=1}^{n} Item_{NotAsInCompDgm}{}^{i}}{\sum_{i=1}^{n} Item_{RequiredInCM}{}^{i}}$$

수식 13 컴포넌트 모델 준수성 측정을 위한 내부 메트릭

$\sum_{i=1}^{n} Item_{NotAsInCompDgm}{}^{i}$ 은 컴포넌트 다이어그램을 통하여 설계 내용에 반영되지 않은 컴포넌트 참조 모델에서의 항목의 개수이며, $\sum_{i=1}^{n} Item_{RequiredInCM}{}^{i}$ 은 표준으로 사용된 컴포넌트 참조 모델에서 제시된 항목의 개수이다. 결과값이 1에 가까울수록 컴포넌트 참조모델에서 제시한 여러 가지의 구성 요소 및 장치들이 컴포넌트 다이어그램의 설계 내용으로 충분히 반영되었음을 의미한다.

6. C-QM의 외부 메트릭

외부 메트릭은 개발이 완료되어 수행 가능한 소프트웨어 제품들을 대상으로 하므로 5장에서의 내부 메트릭의 적용 방법 및 대상과는 다르다. 본 장에서는 4장에서 제시한 소프트웨어 컴포넌트의 품질 측정을 위한 네 가지 품질 요소를 측정하기 위하여 각각의 하위 품질 항목에 적용시킬 수 있는 품질 메트릭들을 제시한다.

6.1 적절성(Suitability)

적절성을 측정하기 위해 사용되는 외부 메트릭은 전체 요구사항을 통해 암시 또는 명시된 기능들을 개수와 구현된 컴포넌트가 가지는 기능들의 개수를 비교함으로써 판단할 수 있다.

$$SuitabilityExt_{FamilyReqSpec}(STRQ) = 1 - \frac{\sum_{i=1}^{n} Fn_{NotImplAsInComp}{}^{i}}{\sum_{i=1}^{n} Fn_{DescribedInFamilyReqSpec}{}^{i}}$$

수식 14 패밀리 요구사항 명세서에 대한 적절성 측정을 위한 외부 메트릭

$\sum_{i=1}^{n} Fn_{NotImplAsInComp}{}^{i}$ 은 개발 완료된 컴포넌트가 가지는 기능들 중 패밀리 요구사항 명세서를 통하여 암시 또는 명시되었지만 구현되지 않은 기능들의 개수이며, $\sum_{i=1}^{n} Fn_{DescribedInFamilyReqSpec}{}^{i}$ 은 패밀리 요구사항 명세서를 통하여 암시 또는 명시되어 있는 전체 기능들의 개수이다. 결과값이 1에 가까울수록 패밀리 요구사항 명세서에 제시된 기능들이 구현된 컴포넌트의 기능으로도 반영이 되었음을 의미한다.

$$SuitabilityExt_{ExtendedFamilyReqSpec}(STER) = 1 - \frac{\sum_{i=1}^{n} Fn_{NotImplAsInComp}{}^{i}}{\sum_{i=1}^{n} Fn_{ModifiedOrExtendedInFamilyReqSpec}{}^{i}}$$

수식 15 확장된 패밀리 요구사항에 대한 적절성 측정을 위한 외부 메트릭

$\sum_{i=1}^{n} Fn_{NotImpleAsComp}{}^{i}$ 은 개발 완료된 컴포넌트가 가지는 기능들 중에 기능상 변경 및 확장된 기능들 중 구현되지 않은 기능들의 개수이며, $\sum_{i=1}^{n} Fn_{ModifiedOrExtendedInFamilyReqSpec}{}^{i}$ 은 패밀리 요구사항의 기능들 중 기능상 변경 또는 확장된 기능들의 개수이다. 결과값이 1에 가까울수록 변경 및 확장된 요구사항의 기능들이 모두 컴포넌트의 기능들로 구현이 되었음을 의미한다.

6.2 완전성(Completeness)

외부 품질을 이용한 외부 메트릭 중 완전성을 측정하기 위한 메트릭은 구현된 컴포넌트가 가지는 전체 기능들 중에서 기능의 일부분이 구현되지 않은 즉, 완전히 구현되지 않은 기능들을 비교함으로써 측정할 수 있다.

$$CompletenessExt(CTEXT) = 1 - \frac{\sum_{i=1}^{n} Fn_{ImplAsInComp}\,i}{\sum_{i=1}^{n} Fn_{ImplAsInComp}\,i}$$

수식 16 완전성 측정을 위한 외부 메트릭

$\sum_{i=1}^{n} Fn_{ImplAsInComp}\,i$ 은 구현된 컴포넌트가 가지는 기능들 중에 기능의 일부분이 구현되지 않은 즉, 완전히 구현되지 않은 기능들의 개수이며, $\sum_{i=1}^{n} Fn_{ImplAsInComp}\,i$ 은 구현된 컴포넌트가 가지는 기능들의 총 개수이다. 결과값이 1에 가까울수록 구현된 기능들 전체가 세부적인 기능까지도 모두 구현되었음을 의미한다. 컴포넌트에 있어서 하나의 독립된 기능으로써 인식되는 기능들과 그 기능을 이루는 세부 기능과의 크기에 있어서 차이를 두어야 한다.

6.3 기능 공통성(Functional Commonality)

C-QM의 기능 공통성을 측정하기 위한 내부 메트릭과는 달리 본 절에서는 수행 가능한 개발 완료된 컴포넌트의 기능을 대상으로 한다. 전체 요구사항 각각의 기능들이 구현된 컴포넌트의 기능으로 충분히 반영되었는지를 측정한다.

$$FunctionalCommonalityExt(FCEXT) = 1 - \frac{\frac{1}{n}\left(\sum_{i=1}^{n} Fn_{NotAsInComp}\,i\right)}{\frac{1}{n}\left(\sum_{i=1}^{n} Fn_{ReqSpec}\,i\right)}$$

수식 17 기능 공통성을 위한 외부 메트릭

$\frac{1}{n}\left(\sum_{i=1}^{n} Fn_{NotAsInComp}\,i\right)$ 은 각 요구사항들의 기능 중 공통된 기능들이 추출되어 반영된 패밀리 요구사항으로부터 구현된 컴포넌트 내에 포함되지 않은 기능들의 평균 개수이며, $\frac{1}{n}\left(\sum_{i=1}^{n} Fn_{ReqSpec}\,i\right)$ 은 전체 요구사항들의 집합을 대상으로 하여 각 요구 사항이 가지는 평균 기능들의 개수이다. 결과값이 1에 가까울수록 각 요구사항들의 기능들이 구현된 컴포넌트에도 충분히 반영되어 기능적으로 컴포넌트의 공통성이 높음을 의미한다.

6.4 특화성(Customizability)

특화성은 설계시 Use Case 다이어그램 등을 통하여 가변성이 포함된 기능성들이 개발 완료된 컴포넌트의 기능들에도 반영이 되어 특화가 가능하도록 구현 되었는지를 측정함으로써 판단할 수 있다. 이러한 가변성은 I_R(Require 인터페이스)을 통하여 설정이 가능하다.

$$CustomizabilityExt(CZEXT) = 1 - \frac{\sum_{i=1}^{n} Fn_{VarNotProvidedAsWithI_R}\,i}{\sum_{i=1}^{n} Fn_{VarProvidedAsInUC}\,i}$$

수식 18 특화성 측정을 위한 외부 메트릭

$\sum_{i=1}^{n} Fn_{VarNotProvidedAsWithI_R}\,i$ 은 구현된 컴포넌트가 가지는 기능들 중에 IR(Require 인터페이스)을 통하여 가변성을 설정할 수 있도록 구현되지 않은 기능들의 개수이며, $\sum_{i=1}^{n} Fn_{VarProvidedAsInUC}\,i$ 은 Use Case 다이어그램 상에서 각 기능 단위인 Use Case에 가변성이 포함되어 설계된 기능들의 개수이다. 결과값이 1에 가까울수록 Use Case 다이어그램 상에서 가변성을 포함한 기능들이 구현된 컴포넌트에서 Require 인터페이스를 통하여 가변성 설정이 가능함을 의미한다.

6.5 결합성(Coupling)

컴포넌트간의 결합성을 측정하기 위해서는 하나의 독립된 컴포넌트를 이루는 구성 클래스들간의 연관 관계에 대하여 컴포넌트와 컴포넌트 사이에서 구성 클래스를 통한 연관 관계 즉, 하나의 클래스가 다른 컴포넌트에도 중복되어 포함되는지 그리고 한 컴포넌트 내의 클래스와 다른 컴포넌트 내의 클래스 간의 연관 관계로써 측정할 수 있다.

$$CouplingExt(CPEXT) = \frac{\sum_{i=1}^{n} ClassRel_{AmongComp}\,i}{\sum_{i=1}^{n} ClassRel_{AsInComp}\,i}$$

수식 19 결합성 측정을 위한 외부 메트릭

$\sum_{i=1}^{n} ClassRel_{AmongComp}\,i$ 은 구현된 컴포넌트들 사이에 존재하는 클래스들 간의 관계들의 개수로서 서로 다른 컴포넌트 내에 존재하는 클래스들간 연관 관계의 개수이며, $\sum_{i=1}^{n} ClassRel_{AmongComp}\,i$ 은 구현된 컴포넌트 내에 존재하는 클래스들 간의 연관 관계들의 총 개수이다. 결과값이 작을수록 컴포넌트 내부의 클래스들 간에 존재하는 연관 관계의 개수에 비해서 동일한 클래스들의 일부가 다른 컴포넌트에 존재함으로써 컴포넌트 간의 상호작용도 높아짐을 의미한다.

6.6 명세 완전성(Specification Completeness)

명세 완전성은 소프트웨어 컴포넌트의 기능들에 대하여 명세서에 얼마나 반영하였는지 측정함으로써 이후의 재사용에 있어서의 용이성을 판단하기 위한 품질 항목이다. 본 논문에서는 크게 두 가지 종류의 명세서 즉, 컴포넌트 명세서와 컴포넌트 인터페이스 명세서에 대하여 명세 정도를 측정한다.

$$SpecificationCompleteness_{CompSpec}(SCCS) = 1 - \frac{\sum_{i=1}^{n} Comp_{NotDescribedInCompSpec}\,i}{\sum_{i=1}^{n} Comp_{Implemented}\,i}$$

수식 20 컴포넌트 명세서의 명세 완전성을 위한 외부 메트릭

$\sum_{i=1}^{n} Comp_{NotDescribedInCompSpec}\,i$ 은 컴포넌트 명세서를 통해서 해당 컴포넌트에 대한 기술이 이루어지지 않은 컴포넌트의 개수이며, $\sum_{i=1}^{n} Comp_{Implemented}\,i$ 은 패밀리 요구사항으로부터 구현된 컴포넌트의 총 개수이다. 결과값이 1에 가까울수록 구현된 컴포넌트들에 대하여 컴포넌트 명세서를 통하여 명세가 잘 이루어졌음을 의미한다.

$$SpecificationCompleteness_{CompIntfSpec}(SCCI) = 1 - \frac{\sum_{i=1}^{n} (I_P)_{NotDescribedInCompIntfSpec}\,i + \sum_{i=1}^{n} (I_R)_{NotDescribedInCompIntfSpec}\,i}{\sum_{i=1}^{n} (I_P)_{Implemented}\,i + \sum_{i=1}^{n} (I_R)_{Implemented}\,i}$$

수식 21 컴포넌트 인터페이스 명세서의 명세 완전성을 위한 외부 메트릭

$\sum_{i=1}^{n} (I_P)_{NotDescribedInCompIntfSpec}\,i$ 은 구현은 되었지만 컴포넌트 인터페이스 명세서를 통하여 명세가 이루어지지 않은 I_P(Provide 인터페이스)의 개수이며, $\sum_{i=1}^{n} (I_R)_{NotDescribedInCompIntfSpec}\,i$ 은 구현은 되었지만 컴포넌트 인터페이스 명세서를 통하여 명세가 이루어지지 않은 I_R(Require 인터페이스[20])의 개수이고, $\sum_{i=1}^{n} (I_P)_{Implemented}\,i$ 은 구현된 컴포넌트가 가지는 I_P(Provide 인터페이스)의 개수이며, $\sum_{i=1}^{n} (I_R)_{Implemented}\,i$ 은 구현된 컴포넌트가 가지는 I_R(Require 인터페이스)의 개수이다. 결과값이 1에 가까울수록 구현된 컴포넌트가 가지는 인터페이스들이 컴포넌트 인터페이스 명세서를 통하여 명세 되었음을 의미한다.

6.7 변경성(Changeability)

변경성을 측정하기 위하여 외부 품질을 이용한 외부 메트릭에 있어서 하나의 기능 단위로 작성되는 시퀀스 다이어그램 상에서 기능들에 대한 변경이 있을 경우 구현된 컴포넌트를 통하여 변경 내용이 충분히 구현되었는지를 측정한다.

$$ChangeabilityExt(CBEXT) = 1 - \frac{\sum_{i=1}^{n} Fn_{UnsuccessfullyChangedInComp}\,i}{\sum_{i=1}^{n} Fn_{ChangedAsInSeq}\,i}$$

수식 22 변경성 측정을 위한 외부 메트릭

$\sum_{i=1}^{n} Fn_{UnsuccessfullyChangedInComp}\,i$ 은 컴포넌트 상에서 설계 내용상의 변경이 적용되지 않아 구현되지 않은 기능들의 개수이며, $\sum_{i=1}^{n} Fn_{ChangedAsInSeq}\,i$ 은 하나의 독립된 기능에 대한 메시지 흐름을 나타내는 시퀀스 다이어그램 상에서 변경이 일어난 기능의 개수이다. 결과값이 1에 가까울수록 시퀀스 다이어그램을 통해서 변경된 기능들이 구현된 컴포넌트에도 잘 반영되어 나타났음을 의미한다.

6.8 단순성(Simplicity)

외부 메트릭을 이용한 단순성은 패밀리 요구사항에서 기능상의 변경을 한 후에 구현된 컴포넌트에도 해당 변경 내용이 충분히 반영되었는지를 측정한다.

$$SimplicityExt(SPEXT) = \frac{\sum_{i=1}^{n} Comp_{Modified}\,i}{\sum_{i=1}^{n} Fn_{ModifiedInFamilyReqSpec}\,i}$$

수식 23 단순성 측정을 위한 외부 메트릭

$\sum_{i=1}^{n} Comp_{Modified}\,i$ 은 기능상의 변경을 위하여 수정된 컴포넌트의 개수이며, $\sum_{i=1}^{n} Fn_{ModifiedInFamilyReqSpec}\,i$ 은 패밀리 요구사항 명세서를 통하여 기능상의 변경을 필요로 하는 기능들의 개수이다. 결과값이 1일 경우는 패밀리 요구사항의 기능 중 평균적으로 하나의 컴포넌트만이 수정을 필요로 하는 것이다.

6.9 도메인 모델 준수성(Domain Model Conformance)

도메인 모델 준수성은 개발된 컴포넌트가 대상으로 하는 도메인에 대하여 해당 도메인 내에서의 주요한 기능들이 충분히 반영되어 구현되었는지를 측정함으로써 준수성을 측정한다.

$$DMConformanceExt(DCEXT) = 1 - \frac{\sum_{i=1}^{n} Fn_{DMRequiredNotAsInComp}\,i}{\sum_{i=1}^{n} Fn_{RequiredInDM}\,i}$$

수식 24 도메인 모델 준수성 측정을 위한 외부 메트릭

$\sum_{i=1}^{n} Fn_{DMRequiredNotAsInComp}i$ 은 구현된 컴포넌트가 가지는 기능들 중 도메인에서 요구되는 주요한 기능들 중 구현되지 않은 기능들의 개수이며, $\sum_{i=1}^{n} Fn_{RequiredInDM}i$ 은 도메인 모델을 통해 요구되는 해당 도메인에서 주요한 기능들의 개수이다. 결과값이 1에 가까울수록 해당 도메인에서 요구되는 주요한 기능들이 구현된 컴포넌트에 충분히 반영되어 구현되었음을 의미한다.

6.10 컴포넌트 모델 준수성(Component Model Conformance)

도메인 모델 준수성과 함께 컴포넌트의 준수성을 측정하기 위한 항목으로써 컴포넌트 참조 모델에 제시된 구현되어야 할 표준 항목들이 개발된 컴포넌트를 통하여 반영되었는지를 측정한다.

$$CMConformanceExt(CCEXT) = 1 - \frac{\sum_{i=1}^{n} Item_{NotAsInComp}i}{\sum_{i=1}^{n} Item_{RequiredInCM}i}$$

수식 25 컴포넌트 모델 준수성 측정을 위한 외부 메트릭

$\sum_{i=1}^{n} Item_{NotAsInComp}i$ 은 컴포넌트 참조 모델의 구성 요소 중 구현된 컴포넌트에 반영되지 않은 항목들의 개수이며, $\sum_{i=1}^{n} Item_{RequiredInCM}i$ 은 표준으로 사용된 컴포넌트 참조 모델에서 제시된 컴포넌트 구현 시 반영되어야 할 항목의 개수이다. 결과값이 1에 가까울수록 표준으로 이용된 컴포넌트 참조 모델에서 요구된 구성 요소들이 구현된 컴포넌트를 통해 충분히 반영되었음을 의미한다.

7. 내부 메트릭과 외부 메트릭 간의 매핑

7.1 외부 메트릭에 대한 영향 요소

STRQ: 개발 완료된 컴포넌트가 가지는 기능들 중 패밀리 요구사항 명세서를 통하여 암시 또는 명시된 기능들이 얼마나 구현되었는지를 측정하는 메트릭으로써, 패밀리 요구사항과 Use Case 다이어그램간의 관계, Use Case 다이어그램 상의 기능과 클래스 다이어그램 상의 클래스 오퍼레이션 간의 일관성, Use Case 다이어그램, 클래스 다이어그램 및 시퀀스 다이어그램들 간의 완전성 및 패밀리 요구사항의 기능 공통성 등으로부터 패밀리 요구사항의 적절성에 대한 외부 품질을 측정한다.

STER: 패밀리 요구사항에 대한 기능상의 확장 및 변경이 반영되었는지를 측정하는 메트릭으로써 패밀리 요구사항과 클래스 오퍼레이션간의 적절성, Use Case 다이어그램, 클래스 다이어그램 및 시퀀스 다이어그램들 간의 완전성, 패밀리 요구사항의 기능적인 공통성 등을 통하여 확장된 패밀리 요구사항에 대한 적절성을 측정한다.

CTEXT: 구현된 기능에 대한 완전성을 측정하기 위한 메트릭으로써, 패밀리 요구사항과 Use Case 다이어그램 간의 적절성, 패밀리 요구사항과 클래스 오퍼레이션간의 적절성, Use Case 다이어그램, 클래스 다이어그램간 및 시퀀스 다이어그램들 간의 완전성 및 Use Case 다이어그램 상에 가변성을 포함하여 설계된 기능들을 측정하는 특화성 등으로부터 측정된다.

FCEXT: 전체 패밀리 멤버들이 가지는 요구사항들이 공통적으로 추출되어 반영되었는지를 측정하는 메트릭으로써, 패밀리 요구사항과 Use Case 다이어그램 간의 적절성, 패밀리 요구사항의 공통 기능에 대한 기능 공통성 및 동일한 도메인 내에서 공통적으로 제시되는 기능들에 대한 도메인 모델 준수성을 이용하여 측정한다.

CZEXT: 컴포넌트의 Require 인터페이스를 통하여 가변성에 대한 구현이 이루어질 수 있는 정도를 측정하는 메트릭으로써, 패밀리 요구사항과 Use Case 다이어그램 간의 적절성 및 Use Case 다이어그램을 통한 가변성 설계 정도에 대한 특화성 등으로부터 측정한다.

CPEXT: 컴포넌트와 내부의 클래스들간의 관계를 이용하여 결합성을 측정하는 메트릭으로써, 해당 도메인에 대한 도메인 모델 준수성과 컴포넌트 내의 클래스들간의 관계를 측정한 결합성을 이용하여 측정한다.

SCCS: 컴포넌트 명세서의 명세 완전성을 위한 메트릭으로써, Use Case 다이어그램, 클래스 다이어그램 및 시퀀스 다이어그램 등에 대한 명세 완전성을 이용하여 측정한다.

SCCI: 컴포넌트 인터페이스의 명세 완전성을 위한 메트릭으로써, Use Case 다이어그램, 클래스 다이어그램 및 시퀀스 다이어그램 등에 대한 명세 완전성을 이용하여 측정한다.

CBEXT: 컴포넌트의 설계 내용상의 변경이 구현 되었는지를 측정하는 것으로써, 패밀리 요구사항 명세서와 소스 코드 상의 구현 관계에 대한 변경성, Use Case 다이어그램과 시퀀스 다이어그램, 시퀀스 다이어그램과 클래스 다이어그램 간의 완전성 등을 이용한다.

SPEXT: 패밀리 요구사항으로부터의 변경이 구현된 컴포넌트에 반영된 정도를 측정하는 메트릭으로써, 패밀리 요구사항 명세서와 소스 코드 상의 구현 관계에 대한 변경성, 수정 및 보완을 위해 필요한 컴포넌트의 기

능 단위에 대한 소스 코드와의 관계에 대한 단순성 및 Use Case 다이어그램에 가변성이 식별되어 설계된 기능에 대한 특화성 등을 이용한다.

DCEXT: 대상으로 하는 해당 도메인에서의 주요한 기능들이 충분히 반영되어 구현되었는지를 측정하는 메트릭으로써, 전체 요구사항 집합을 대상으로 하여 공통적인 기능들에 대한 추출 정도인 기능 공통성과 도메인에서 공통적으로 요구되는 기능들에 대한 Use Case로의 식별 여부 등이 이용된다.

CCEXT: 컴포넌트 참조 모델에서 제시하는 표준 항목들에 대한 구현 정도를 측정하는 메트릭으로써, 컴포넌트 다이어그램을 통하여 컴포넌트 참조모델에서 제시한 구성 요소들에 대한 반영 정도가 이용된다.

7.2 매핑 테이블

본 절에서는 5장과 6장을 통해서 도출된 소프트웨어 컴포넌트의 품질 속성 측정을 위한 내부 및 외부 메트릭들이 서로 간에 어떠한 연관 관계를 가지고 있는지 매핑관계를 나타낸다. 7.1에서 제시된 각 외부 메트릭에 대한 영향 요소들이 적용되어 내부 메트릭 각각이 어떠한 외부 메트릭에 대하여 영향을 미치는지를 나타낸다.

7.3 품질 추정 모델

본 절에서는 그림 7에서 매핑된 내부 메트릭과 외부 메트릭 간의 관계를 통하여 품질 항목 및 품질 요소 등을 예측함으로써 최종 소프트웨어 제품이 갖게 될 품질을 추정할 수 있는 모델을 제시한다.

표 5에서 해당 외부 메트릭에 영향을 주는 내부 메트릭의 개수에 따라 차등적으로 값을 곱해주며 이렇게 하여 측정된 외부 메트릭의 값들로써 하나의 외부 품질 요소에 대한 값을 측정하게 된다. 이로써, 내부 메트릭을 적용하여 얻어진 결과값으로써 외부 메트릭을 측정하며 이러한 외부 메트릭 값을 통하여 외부 품질에 있어서의 품질 항목에 대한 값과 최종적인 외부 품질 요소에 대한 값을 측정하게 된다.

추정 모델에 제시된 공식은 그림 7에서 제시한 내부 메트릭과 외부 메트릭 간의 관계를 기반으로 하였으며, C-QM의 각 품질 요소 및 품질 항목들은 범용적인 소프트웨어가 아닌 컴포넌트를 위하여 이미 특화된 모델에 대한 것이므로 각 메트릭과 품질 항목을 이용하여 품질 요소를 계산시 각각의 메트릭에 대한 결과값 및 품질 항목에 대한 값들이 동일하게 계산될 수 있도록 하였다. 예를 들어, 내부 메트릭 5가지의 결과값으로 계산되는 STRQ와 3개의 내부 메트릭 결과값으로 측정되는 FCEXT에 대한 계산 결과가 동일하게 하나의 기능성이라는 품질 요소를 측정하기 위한 하위의 품질 항목들이 되도록 하였다.

결과적으로는 총 4개의 품질 요소들은 소프트웨어 컴포넌트의 품질을 측정하는데 있어서 4가지 요소에 대한 계산값들이 모두 동일하게 측정 및 적용되도록 하였다.

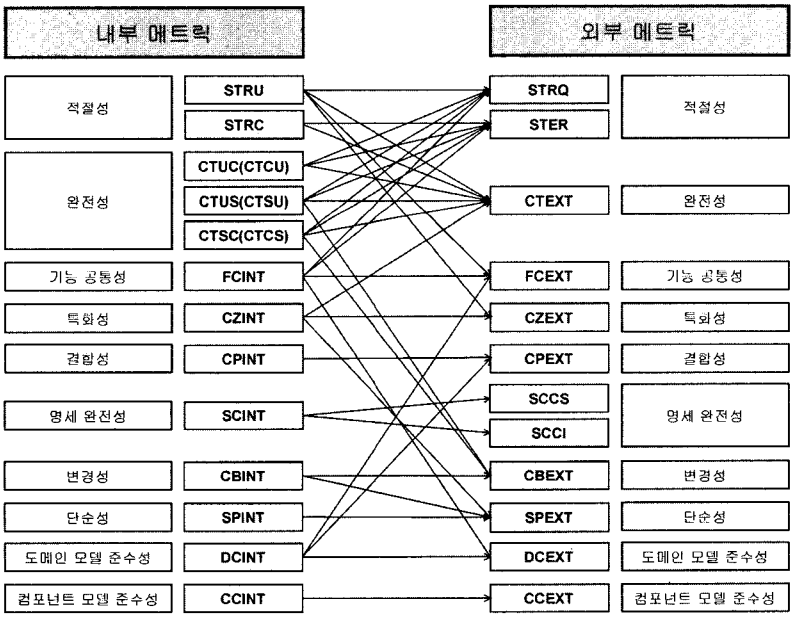

그림 7 내부 메트릭과 외부 메트릭의 관계

표 5 내부 메트릭을 이용한 외부 품질 추정 모델

외부 품질 요소	외부 품질 요소에 대한 추정값	외부 품질 항목	외부 메트릭	관련 내부 메트릭
기능성	{(0.15×STRQ)+(0.15×STER)+(0.3×CTEXT)+(0.3×FCEXT)}	적절성(2)	STRQ(5)	(0.2×STRU)+{0.2×CTUC(CTCU)}+{0.2×CTUS(CTSU)}+{0.2×CTSC(CTCS)}+(0.2×FCINT)
			STER(5)	(0.2×STRC)+(0.2×CTUC)+(0.2×CTUS)+(0.2×CTSC)+(0.2×FCINT)
		완전성(1)	CTEXT(6)	(0.17×STRU)+(0.17×STRC)+(0.17×CTUC)+(0.17×CTUS)+(0.17×CTSC)+(0.17×CZINT)
		기능 공통성(1)	FCEXT(3)	(0.33×STRU)+(0.33×FCINT)+(0.33×DCINT)
재사용성	{(0.3×CZEXT)+(0.3×CPEXT)+(0.15×SCCS)+(0.15×SCCI)}	특화성(1)	CZEXT(2)	(0.5×STRU)+(0.5×CZINT)
		결합성(1)	CPEXT(2)	(0.5×DCINT)+{0.5×(-CPINT)}
		명세 완전성(2)	SCCS(1)	SCINT
			SCCI(1)	SCINT
유지 보수성	{(0.5×CBEXT)+(0.5×SPEXT)}	변경성(1)	CBEXT(3)	(0.3×CBINT)+(0.3×CTUS)+(0.3×CTSC)
		단순성(1)	SPEXT(3)	(0.3×CBINT)+{0.3×(- SPINT)+(0.3×CZINT)
준수성	{(0.5×DCEXT)+(0.5×CCEXT)}	도메인 모델 준수성(1)	DCEXT(2)	(0.5×FCINT)+(0.5×DCINT)
		컴포넌트 모델 준수성(1)	CCEXT(1)	CCINT

8. 사례 연구

본 장에서는 표 5의 외부 품질 추정 모델을 이용하여 소프트웨어 컴포넌트 개발에 따른 산출물에 대하여 내부 메트릭과 외부 메트릭을 적용시키는 예를 보임으로써, 이를 이용한 품질 추정 모델의 유용성과 적합성을 검증한다.

8.1 CBD 산출물

다음 절에서는 본 논문에서 제시된 내부 메트릭 및 외부 메트릭이 적용될 수 있는 소프트웨어 컴포넌트 개발 산출물들을 알아본다. 본 논문에서는 이를 위하여 '병원관리시스템(HMS)'을 도메인으로 하여 다음과 같은 산출물들에 적용시켜 본다.

<A 병원>

A 병원은 병원 관리 시스템 개발을 계획하고 있다. A 병원은 진찰을 받기 전에 예약을 해야 한다. 예약을 접수할 때에는 예약 접수원이 환자의 이름, 환자의 주민등록번호, 환자가 진찰을 받고 싶은 날짜, 시간(오전, 오후), 진찰 받고자 하는 진찰과 등의 정보를 환자에게 듣고 예약을 접수한다. 이때 진찰을 받기위한 예약을 접수하러 온 환자가 A병원에 등록된 환자인지를 검사하여 등록되지 않은 환자이면 환자에게서 환자의 이름, 주민등록번호, 주소 등의 간략한 정보를 받아서 환자를 접수한다. 환자 접수시 환자에게 받은 주민등록번호를 이용하여, 그 값으로써 환자에 대한 고유 ID를 부여한다. 또한, A병원에서는 각 시간대별(오전, 오후)로 한 의사가 진찰할 수 있는 환자의 수가 10명으로 제한 되어있다. 따라서 예약을 접수할 때, 이 조건을 감안하여 의사를 배정한다. 환자는 예약을 변경할 수 있다. 변경하고 싶은 정보(진찰 받고 싶은 날짜, 시간, 진찰 받고자 하는 진찰과)를 예약 접수원에게 말하여 변경이 가능하면 예약을 변경하게 된다. 또한 환자는 예약을 취소할 수가 있다. 환자는 병원의 자동응답 서비스를 이용하여 환자의 주민등록번호 숫자 13자리를 누르면 자신의 예약에 대한 정보를 확인할 수가 있다.

예약된 정보를 가지고 하루 치료 일정표를 만들어 의사가 진찰할 때에 사용하게 된다. 의사는 의사 별로 작성된 하루 치료 일정표를 보고, 그날의 환자를 진찰한다. 환자를 진찰할 때 환자의 병력을 기록하고, 처방을 내린다.

<B 병원>

B 병원은 병원 관리 시스템을 개발하려 한다. 이 병원에서는 진료를 받기 위해 예약을 할 수가 있다. 예약을 등록할 때에는 예약 담당 관리자가 환자의 이름, 환자의 주민등록번호, 환자가 진료를 받고 싶은 날짜, 시간(오전, 오후), 진료 받고 싶은 진료과목 등의 정보를 환자에게 받아서 예약을 등록한다. 이때 진찰을 받기 위한 예약을 접수하러 온 환자가 A병원에 등록된 환자인지를 검사하여 등록되지 않은 환자이면 환자에게서 환자의 이름, 주민등록번호, 주소, email주소 등의 간략한 정보를 받아서 환자를 접수한다. 환자 접수시 환자에 대한 일련의 순차적인 고유번호를 정한 후, 그 번호로써 환자에 대한 고유 ID를 부여한다. 또한, 이 병원에서는 각 시간대별(오전, 오후)로 한 의사는 10명 이하의 환자만을 진료할 수가 있다. 예약을 등록할 때, 이런 예약 가능 조건을 검사하여 예약을 등록한다. 환자는 예약을 변경할 수가 있다. 변경하고 싶은 정보(진료 받고 싶은 날짜, 시간, 진료 받고자 하는 진료과목)에 대해 예약 담당 관리자를 통하여 예약을 변경한다. 또한 환자는 예약을 취소할 수가 있다. 이 병원에서는 예약등록, 변경, 취소에 대한 정보를 email을 통해 환자에게 통보하게 된다.

환자에 대한 예약된 정보를 가지고 일일 진료 스케줄을 만든다. 의사는 의사 별로 작성된 일일 진료 스케줄을 보고, 그날의 환자를 진료한다. 환자를 진료할 때 환자의 병명을 기록하고, 처방전을 내린다.

그림 8 병원 A와 병원 B에 대한 요구사항

<R1. 예약 등록>
진료를 받기 전에 예약을 해야한다. 예약을 등록할 때에는 예약 관리자가 환자의 이름, 환자의 주민등록번호, 환자가 희망하는 진료 날짜, 시간(오전, 오후), 진료받고자 하는 진료과목등의 정보를 환자에게 받고난 후 예약을 등록한다. 이때 진료를 받기위한 예약을 등록하러 온 환자가 병원에 등록된 환자인지를 검사하여 등록되지 않은 환자이면 환자 등록을 한다. 예약을 등록할 때에는, 예약 가능 조건을 검사하여 예약을 등록한다.

<R2. 환자 등록>
환자에게서 환자의 이름, 주민등록번호, 주소,(Optional: email주소)등의 간략한 정보를 받아서 환자를 등록한다. 환자등록시 환자의 입력 받은 주민등록번호(또는, 일련의 순차적인 고유번호)를 이용하여, 그 값으로써 환자에 대한 고유 ID를 생성한다.

<R3. 예약 가능 조건 검사>
각 시간대별(오전, 오후)로 한 의사는 10명이하의 환자만을 진료할 수 있다. 따라서 각 시간대별(오전, 오후)로 진료과목에 대한 예약 환자수가 10명이만인 의사만 예약가능하다.

<R4. 예약 변경>
환자는 예약 변경 가능하다. 변경하고 싶은 정보(진료 희망 날짜, 시간, 진료받고자 하는 진료과목)를 예약 관리자에게 알하여 변경이 가능하면 예약을 변경하게 된다.

<R5. 예약 취소>
환자는 예약취소 가능하다.

R6. 예약 정보 email로 전송
예약등록, 변경, 취소에 대한 정보를 email을 통해 환자에게 통보하게 된다.(Optional)

<R7. 일일 진료 일정표 만들기>
환자에 대한 예약된 정보를 가지고 일일 진료 일정표를 작성한다.

<R8. 일일 진료 일정표 보기>
의사는 의사별로 작성된 일일 진료 일정표를 보고, 그날의 환자를 진료한다.

<R9. 환자의 병력 보기>
의사가 환자를 진료할 때 기존의 병력을 참조한다.

<R10. 환자의 병명과 처방 기록하기>
환자를 진료할 때 환자의 병명을 기록하고, 처방을 내린다.

그림 9 HMS에 대한 패밀리 요구사항

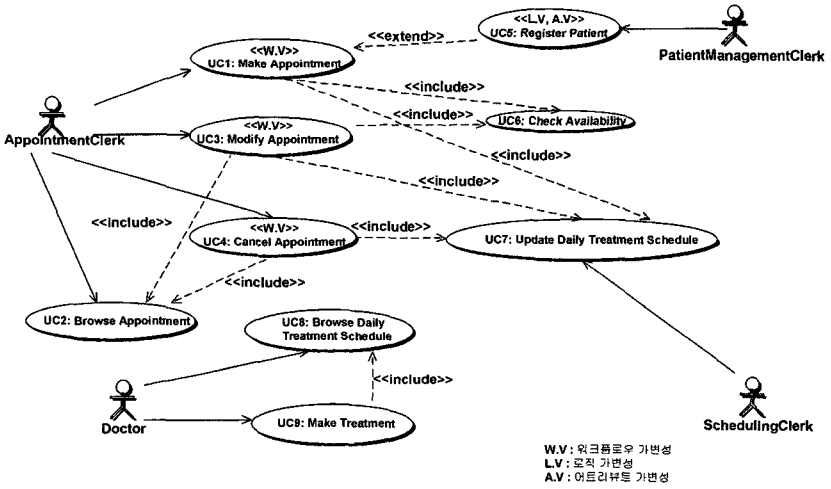

그림 10 HMS에 대한 Use Case 다이어그램

두 개의 병원 A와 B 각각에 대한 다음과 같은 요구사항이 존재한다. 병원 A와 B 사이에는 공통된 기능들 뿐 아니라 동일한 기능에 대하여 차이점도 가지고 있다.

두 개의 요구사항으로부터 추출된 공통된 기능들에 대한 패밀리 요구사항 명세는 다음과 같다. 10가지의 독립적인 기능들로 구성되어 있으며 이들 중 몇 가지는 서로 다른 기능들에 대하여 설정 할 수 있는 가변성을 포함하고 있다.

그림 10에서는 HMS의 Use Case 다이어그램을 나타낸다. 전체 9개의 Use Case 들로 구성되며 이 중 네 개의 Use Case는 각각 가변성을 포함하고 있는 기능들을 나타낸다.

그림 11에서는 Use Case 다이어그램 상에 나타난 각각의 Use Case에 대하여 주요흐름, 선택흐름, 예외흐름 및 시나리오 등으로 이루어진 Use Case Description을 나타낸다.

그림 12에서는 HMS의 주요 클래스들로 구성된 클래스 다이어그램을 나타낸다. 환자, 의사, 예약, 진료스케줄, 진료 및 예약 컨트롤러 객체 등을 나타내고 있다.

그림 13에서는 '예약하다' 라는 기능에 대한 시스템 시퀀스 다이어그램을 나타낸다. HMS에 대하여 Actor가 사용하는 오퍼레이션 및 HMS로부터의 오퍼레이션

< UC1: Make Appointment>

O Actor

AppointmentClerk

O Overview

진료를 원하는 환자는 AppointmentClerk를 통하여 예약을 등록한다.

< Alternative Flow>

A-1

환자 정보가 없으면 'Register Patient' use case를 호출한다.

< Exception Flow>

E-1

Check Availability의 검사 결과, 유효한 예약날짜와 시간이 없으면 "예약이 가능하지 않습니다. 날짜와 시간을 다시 정하십시오"라는 에러 메시지를 보인다.

< Scenario>

환자 "장수"는 진료예약을 요구한다.

Appointment Clerk "길동" 은 진료예약 메뉴를 선택한다.

화면에 진료 예약 정보 항목이 보여진다.

Appointment Clerk "길동" 은 진료 예약 정보 항목인 환자 성명 "장수", 주민등록번호 "771111-1234567", 진료과목 "내과", 예약날짜 "2001-10-5", 예약 시간(오전, 오후) "오전" 을 입력하고 확인 버튼을 누른다.

예약이 성공적으로 처리되었습니다. 라는 메시지가 화면에 출력된다.

< Main Flow>

Actor	System
1. Make Appointment을 요청한다.	
	2. 진료 예약 폼을 출력한다. - 예약 항목 : 성명, 주민등록번호, 진료과목, 예약날짜, 예약시간(오전, 오후)
3. 진료 예약 폼에 환자의 성명, 주민등록번호, 진료과목, 예약날짜, 예약시간 (오전, 오후)를 입력한다.	
	4. 환자의 등록 여부를 확인한다. (A-1)
	5. 'Check Availability' use case를 호출한다.(E-1)
	6. 진료과목에 대해 예약 가능한 의사를 배정한다.
	7. 'Update Daily Treatment Schedule'을 호출한다.
	8. 예약 정보를 저장한다.
	9. < < W.V: Optional> > 환자에게 예약 등록에 대한 정보를 email로 전송 (Default : 추가)
	10. 예약 등록이 성공적으로 이루어졌음을 알리는 메시지를 출력한다.
	11. Make Appointment 을 종료한다.

그림 11 '예약하다'에 대한 Use Case Description

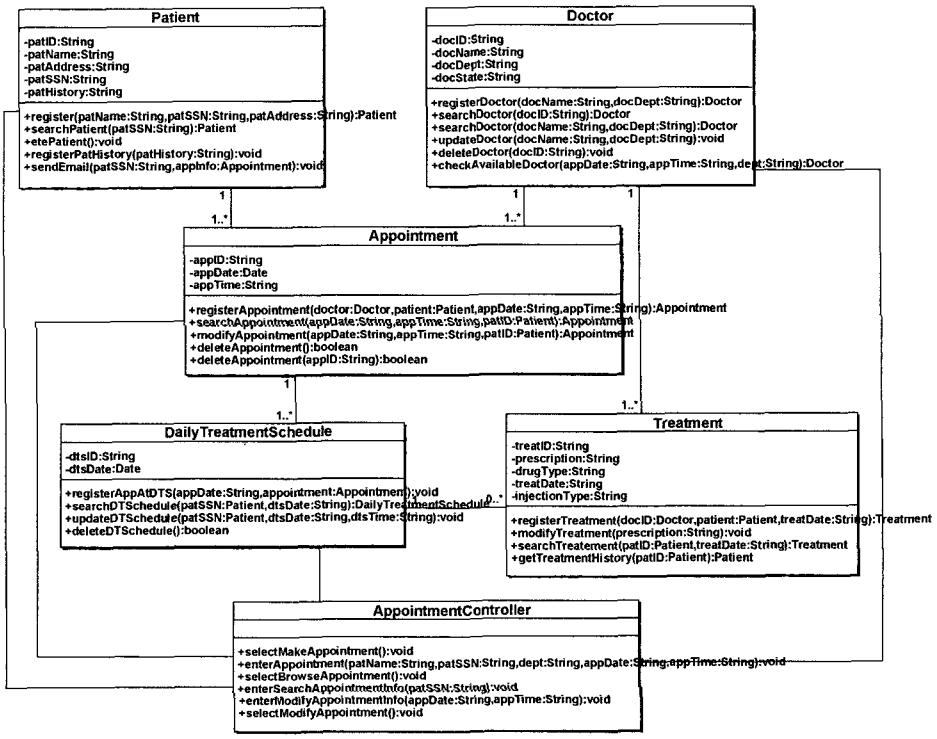

그림 12 HMS의 클래스 다이어그램

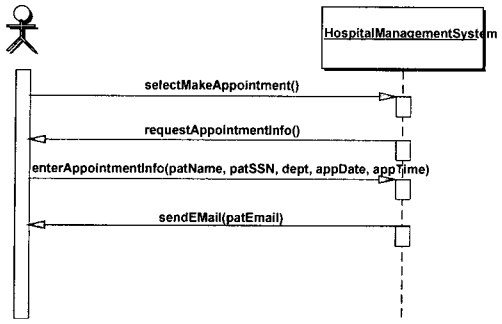

그림 13 '예약하다'에 대한 시스템 시퀀스 다이어그램

등을 나타내고 있다.

그림 14에서는 '예약하다' 라는 Use Case에 대하여 참여하는 객체들간의 메시지 흐름을 나타낸다. 객체들간의 메시지의 흐름에도 가변성이 포함된 경우에는 표시가 되어있다.

그림 15에서는 '예약'기능과 '진료'에 대한 기능을 가지고 있는 컴포넌트에 대한 컴포넌트 다이어그램을 나타낸다. 컴포넌트를 구성하는 내부 클래스들과 그들간의 관계, 그리고 관련된 Use Case 다이어그램 등이 함께 나타나게 된다.

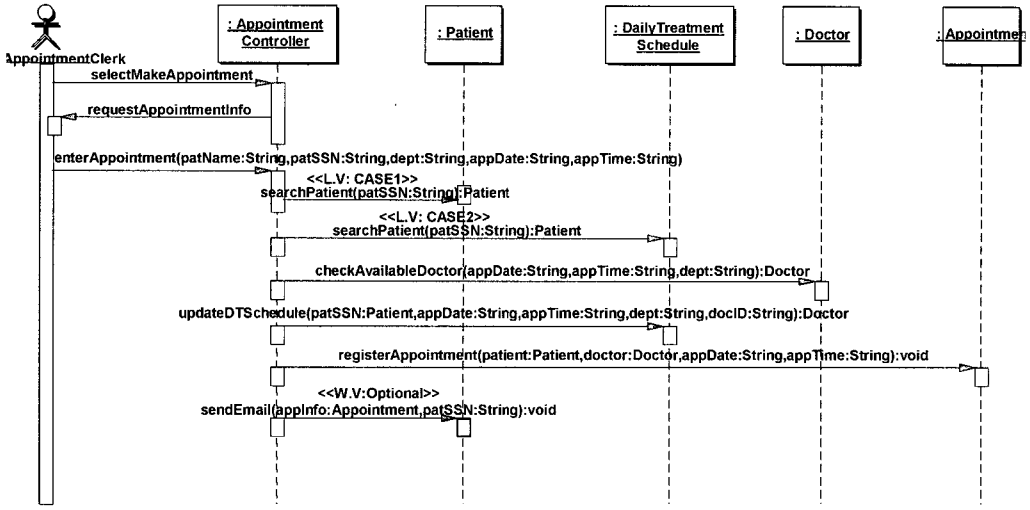

그림 14 '예약하다'에 대한 시퀀스 다이어그램

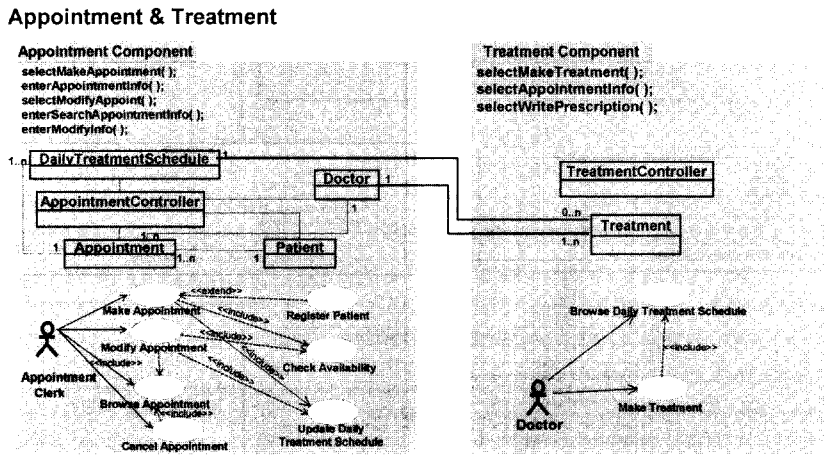

그림 15 '예약' 및 '진료'에 대한 컴포넌트 다이어그램

8.2 내부 메트릭 적용의 예

본 절에서는 8.1에 나타낸 CBD 각 산출물들에 대하여 5장에서 제시한 내부 메트릭을 적용한 결과값을 나타낸다.

표 6 내부 메트릭 적용 결과

내부메트릭	분모값	분자값	측정값
STRU	10	1	**0.90**
STRC	10	1	**0.90**
CTUC	9	0	**1.00**
CTUS	9	1	**0.90**
CTSC	30	0	**1.00**
FCINT	(10+9)/2	(1+0)/2	**0.95**
CZINT	9	5	**0.44**
CPINT	21	6	**0.29**
SCINT	9+6+3	0	**1.00**
CBINT	3	0	**1.00**
SPINT	8	3	**2.67**
DCINT	18	16	**0.11**
CCINT	7	0	**1.00**

8.3 외부 메트릭 적용의 예

본 절에서는 본문에서 제시된 외부 메트릭을 이용하여 외부 품질 항목을 측정하게 된다. 8.2에서 나타낸 내부 메트릭을 이용한 외부 메트릭 값의 추정이 아닌 소프트웨어 컴포넌트가 개발된 이후에 외부 메트릭을 적용시켜 측정된 값들이다.

표 7 외부 메트릭 적용 결과

외부메트릭	분모값	분자값	측정값
STRQ	10	1	**0.90**
STER	5	1	**0.80**
CTEXT	9	1	**0.89**
FCEXT	(10+9)/2	(1+0)/2	**0.95**
CZEXT	4	0	**1.00**
CPEXT	21	6	**0.29**
SCCS	4	0	**1.00**
SCCI	4+5	0	**1.00**
CBEXT	5	1	**0.80**
SPEXT	4	2	**0.50**
DCEXT	18	16	**0.89**
CCEXT	7	0	**1.00**

8.4 매핑 결과

본 절에서는 8.2와 8.3을 통하여 내부 메트릭 및 외부 메트릭을 적용한 결과를 가지고 외부 품질을 추정하며 또한 추정된 값과 외부 메트릭을 이용하여 측정된 값 사이의 관계를 알아본다.

표 8에서는 내부 메트릭을 이용하여 추정된 외부 품질과 외부 메트릭을 이용하여 직접 측정된 결과값을 비교함으로써 제시된 내부 및 외부 메트릭 간의 관계를 나타내고 있다. 측정하고자 하는 외부 품질 요소인 기능성, 재사용성, 유지보수성, 준수성 등은 내부 메트릭으로 측정된 결과값을 이용하여 표 5에서 제시된 공식에 따

표 8 내부 메트릭 이용한 추정값과 외부 메트릭의 매핑

외부 메트릭	내부 메트릭 이용한 추정값	외부 메트릭 이용한 측정값	외부 품질 항목	외부 품질 요소		외부 메트릭 이용한 측정값
STRQ	**0.95**	**0.90**	적절성	기능성	**0.74**	**0.82**
STER	**0.95**	**0.80**				
CTEXT	**0.86**	**0.89**	완전성			
FCEXT	**0.65**	**0.95**	기능 공통성			
CZEXT	**0.96**	**1.00**	특화성	재사용성	**0.31**	**0.51**
CPEXT	**0.09**	**0.29**	결합성			
SCCS	**1.00**	**1.00**	명세완전성			
SCCI	**1.00**	**1.00**				
CBEXT	**0.68**	**0.80**	변경성	유지 보수성	**0.11**	**0.15**
SPEXT	**0.46**	**0.50**	단순성			
DCEXT	**0.49**	**0.89**	도메인모델 준수성	준수성	**0.75**	**0.95**
CCEXT	**1.00**	**1.00**	컴포넌트모델 준수성			

라 계산한 추정값 및 측정값 들을 각 품질 요소에 따라 비교한다. 최종적으로 이러한 결과값들 즉, 내부 메트릭을 이용한 추정값과 외부 메트릭을 이용하여 측정된 값들 사이의 관계 정도를 상관분석을 이용하여 파악한다.

8.5 추정값과 측정값의 상관관계 분석

8.4에서 제시된 내부 메트릭을 이용한 추정값과 외부 메트릭을 이용하여 측정된 값 사이의 상관관계를 분석한다. 메트릭의 적용 대상으로 이용된 HMS는 하나의 단일 도메인이며, 이러한 여러 가지 도메인에 대하여 반복적인 메트릭의 적용을 통한 결과값으로부터 상관관계를 도출해야 하지만 현실적으로 모든 도메인에 대한 적용을 직접 하지 않고 본 논문에서 적용된 HMS에 대한 결과값을 표본으로 이용하는 표본 상관계수(γ)를 통하여 분석한다. 표본 상관계수는 다음과 같은 공식을 이용하여 계산되며 결과값은 표 9와 같다.

$$\gamma = \frac{\sum (X_\phi - \bar{X})(Y_\phi - \bar{Y})}{\sqrt{\sum (X_\phi - \bar{X})^2 \sum (Y_\phi - \bar{Y})^2}}$$

표 9 표본 상관계수

	내부 메트릭을 이용한 추정값 (X)	외부 메트릭을 이용한 측정값 (Y)
기능성	0.74	0.82
재사용성	0.31	0.51
유지보수성	0.11	0.15
준수성	0.75	0.95
γ = 0.976154		

표 9의 결과에 따라 내부 메트릭을 이용한 추정값과 외부 메트릭을 이용한 측정값들 간의 선형적인 관계가 있음을 알 수 있다. 표본 상관계수의 결과값은 +1.0과 -1.0 사이의 값을 갖는데 ±1.0에 가까울수록 서로간에 관계가 높음을 의미하며, 계산값이 0에 가까울수록 서로간의 관계가 적음을 의미한다. 표본 상관계수 값은 메트릭의 결과값 자체에 대한 상관관계를 나타내며, 이는 내부 메트릭으로 측정된 결과값을 이용하여 추정된 외부 품질 요소들은 외부 메트릭으로 측정된 값들에 영향을 주는 요소가 된다. 따라서, 개발이 완료되기 이전에 내부 메트릭을 이용하여 추정된 외부 품질 요소의 값들은 본 논문에서 제시한 관계에 따라 영향을 미치는 요소들이 되며, 각 품질 요소에 대한 품질 항목들에 대하여 추정값을 이용하여 개발 완료 이전에 품질을 예측 및 관리할 수 있다.

9. 결 론

재사용성 및 범용성의 장점을 이용한 컴포넌트 기반의 개발에서는 컴포넌트의 품질 자체가 전체 소프트웨어 개발에서 큰 비중을 차지하므로, 고품질의 컴포넌트를 개발함으로써 컴포넌트의 재개발 등에 소요되는 비용을 줄일 수 있다. 하지만, 컴포넌트 자체의 품질에 대한 연구가 미흡하며 범용적인 소프트웨어에 적용될 수 있는 품질 모델이 컴포넌트 소프트웨어에 적용되기에도 미흡하다. 따라서, 본 논문에서는 컴포넌트에 특화 시켜 적용할 수 있는 소프트웨어 컴포넌트를 위한 품질과 품질 모델, 이들 간의 관계를 통한 적용 방법들을 제시한다.

기본적인 소프트웨어 컴포넌트의 특징들에 기반하여 컴포넌트를 위한 품질 모델을 제시하고, 컴포넌트 개발 과정에 있어서의 중간 산출물 및 개발 완료 후에 적용시킬 수 있는 내부 및 외부 메트릭들을 제시한다. 마지막으로 이들 간의 관계 및 CBD 산출물 각각에 적용시킨 결과를 이용하여 컴포넌트의 내부 품질 및 외부 품질을 비교 및 추정할 수 있도록 한다. 이를 이용하여 소프트웨어 컴포넌트를 개발하는데 있어서 개발 도중에 내부 메트릭을 적용 함으로써 개발 완료 후의 고품질의 컴포넌트 개발을 유도할 수 있다. 따라서, 소프트웨어 개발 전체에 소요되는 비용을 절감할 수 있다.

참 고 문 헌

[1] IEEE Std., 1061, *Software Quality Metrics Methodology*, 1998.

[2] Schulmeyer, G. G. and McManus, J. I., *Handbook of Software Quality Assurance Third Edition*, Prentice Hall PTR, 1999.

[3] Lerouge, C., Garfield, M. J. and Hevner, A. R., "Quality Attributes in Telemedicine Video Conferencing," Proceedings of the 35th Hawaii International Conference on System Science, 2002.

[4] Losavio, F. and Chirinos, L., "Quality Models to Design Software Architectures," *IEEE, 2001.*

[5] Jcaquet, J. and Abran, A., "From Software Metrics to Software Measurement Methods," the Third International Symposium and Forum on Software Engineering Standards, ISESS '97, Walnut Creek (CA), June 2-6, 1997.

[6] Bansiya, J. and Davis, C.G., "A Hierarchical Model for Object-Oriented Design Quality Assessment," IEEE Transactions on Software Engineering, Vol.28, No.1, January 2002.

[7] Briad, L. C., Wust, J., Ikonomovski, S. V. and Lounis, H., "Investigating Quality Factors in Object-Oriented Designs : an Industrial Case Study," ICSE '99 Los Angeles CA, p.345-354, 1999.

[8] Liu, K., Zhou, S. and Yang, H., "Quality Metrics

of Object Oriented Design for Software Deve lopment and Re development," Proceedings of the First Asia Pacific Conference on Quality Software, 2000.
[9] Schneidewind, N. F., "Software Metrics Model for Integrating Quality Control and Prediction," Proceedings of the 8th International Symposium on Software Reliability Engineering, 1997.
[10] Scheneidewind, N. F., "Software Metrics for Quality Control," Proceedings of the 4th Inter national Software Metrics Symposium, p.127 136, 1997.
[11] ISO/IEC, *FCD 9126 1.2 Information Technology -Software product quality -Part 1:Quality model*, 1998.
[12] ISO/IEC, *FCD 9126 3 : Software Engineering - Product quality -Part 3 : Internal Metrics*, 2000.
[13] ISO/IEC, *FCD 9126 2 : Software Engineering - Product quality -part 2 : External metrics*, 2000.
[14] ISO/IEC, *DTR 9126 4 : Software Engineering - Software Product Quality - Part 4 : Quality in Use Metrics*, 2001.
[15] McCall, J. A., *Software Quality Management*, A Petrocelli Book, 1979.
[16] Boehm, B. W., Brown, J. R., Lipow, H., MacLeod, G. J. and Merrit, M. J., *Characteristics of Soft ware Quality*, Elsevier North Holland, 1978.
[17] Dromey, R. G., "A Model for Software Product Quality," IEEE Transactions on Software Engin eering, Vol., 21, No.2, February, 1995.
[18] Kececi, N. and Abran, A., "Analyzing, Measuring & Assessing Software Quality within A Logic Based Graphical Framework," 4e congres Plusi disciplinaire Qualite et surete de fonctionnement Annecy, France, 22 23 mars, 2001.
[19] Seffah, A., Kececi, N. and Donyaee, M., "QUIM : A Framework for Quantifying Usability Metrics in Software Quality Models," IEEE, 2001.
[20] Heineman, G. T. and Councill, W. T., "Component -Based Software Engineering," Addison-Wesley, 2001.
[21] Cai, X., Lyu, M. R., Wong, K. and Ko, R., "Component Based Software Engineering : Tech nologies, Development Frameworks, and Quality Assurance Schemes," Proceedings of the Seventh Asia Pacific Software Engineering Conference, p.372 379, 2000.
[22] Sedigh-Ali, S., Ghafoor, A., Paul, R. A., "Metrics Guided Quality Management for Component Based Software Systems," Proceedings of the 25th Annual International Computer Software and Applications Conference, 2001.
[23] "CORBA Components," Object Management Group, Inc., June, 2002.
[24] Haines, C., Carney, D. and Foreman, J., "Compo nent Based Software Development/COTS Integra tion," *CMU Software Technology Review*, Octo ber, 1997.
[25] "Enterprise JavaBeans Specification, Version 2.1," Sun Microsystems, Inc., 2002.
[26] Perrone, P., *Building Java Enterprise Systems with J2EE*, Sams Publishing, 2000.

박 지 환
정보과학회논문지 : 소프트웨어 및 응용
제 30 권 제 1 호 참조

신 석 규
1995년 3월~1998년 8월 충남대 대학원 컴퓨터과학과. 1979년 10월~1981년 5월 KIST(한국과학기술연구원) 전산개발센터 연구원. 1981년 5월~1985년 8월 SANCST(사우디아라비아국립과학기술연구소) 파견(기술고문). 1985년 8월~1995년 6월 KAIST(한국과학기술원) 시스템공학연구소 선임연구원. 1986년 1월~1988년 12월 '88서울올림픽 전산담당관. 1992년 1월~1993년 12월 '93 대전엑스포 회장운영시스템 개발단장. 1995년 7월~1998년 5월 시스템공학연구소 경영정보과장. 1998년 6월~2001년11월 ETRI(한국전자통신연구원) 정보화지원팀장, 시스템통합연구팀장, S/W시험운영팀장. 2001년 12월~현재 TTA IT시험연구소 S/W시험센터 S/W시험운영팀장

김 수 동
정보과학회논문지 : 소프트웨어 및 응용
제 30 권 제 1 호 참조