결합도 측정 메트릭을 이용한 객체지향적 개발 소프트웨어의 영향도 분석에 관한 연구

권예진*, 박용범**
*단국대학교 전자계산학과
**단국대학교 컴퓨터공학과
e-mail:{kwon6838*, ybpark**}@dankook.ac.kr

Study on the Analysis of the impact of object-oriented software development using the coupling metrics that measure

Yejin Kwon*, Youngbom Park**
*Dept of Computer Science, Dankook University
**Dept of Computer Science, Dankook University

요 약

소프트웨어가 복잡해지고 대형화됨에 따라 다양한 소프트웨어 측정 개발에 필요한 비용이 점차 증가하게 되었지만 지금까지 시도된 측정 기법은 정형화되어 있지 않고 다양한 측정 메트릭을 통해 소프트웨어를 측정한다고 해도 측정 결과의 비교와 분석을 통해 수치적 해석 데이터를 얻기가 어려웠다. 따라서 본 논문에서는 사용자의 사용 사례를 통해 패턴을 추출하고 이를 통해 실험 데이터를 만들어 실제 소프트웨어가 동작하는 도중 내부 모듈간의 동적인 관계를 측정할 수 있는 시스템을 제안하였다.

1. 서론

소프트웨어가 복잡해지고 대형화됨에 따라 다양한 소프트웨어 측정 개발에 필요한 비용이 점차 증가하게 되었다. 객체지향 개발 기법의 중요성을 감안할 때, 소프트웨어의 측정을 통한 복잡도나 품질 요소들의 측정 기술에 대한 요구는 필수불가결한 요소가 되었으며, 현재까지 다양한 측정 기술이 시도되었다. 그러나 지금까지 시도된 측정 기법은 정형화되어 있지 않고 다양한 측정 메트릭을 통해 소프트웨어를 측정한다고 해도 측정 결과의 비교와 분석을 통해 수치적 해석 데이터를 얻기가 어렵다. 이러한 문제점들은 정확한 소프트웨어 측정을 어렵게 하며 소프트웨어의 복잡도의 정확한 분석이 이루어지지 않기 때문에 개발과 유지보수를 더욱 힘들게 한다. 소프트웨어는 자체적인 결함뿐만 아니라 더 좋은 품질을 유지하기 위해 지속적인 유지보수와 기존 소프트웨어의 재사용을 위해 다양한 노력이 필요하다. 따라서 소프트웨어의 정확한 측정과 분석은 전체적인 소프트웨어 개발 프로세스에서 중요한 부분을 차지하게 된다. 소프트웨어의 측정에 사용되는 기법은 다양하지만 본 논문에서는 소프트웨어의 각 모듈 또는 클래스 사이의 영향도를 측정하고 상호간의 의존도와 파급력을 분석하여 기존 소프트웨어의 재사용성이나 유지보수성을 증대시키기 위해 적합한 결합도 측정 방법을 이용하여 소프트웨어 측정을 할 수 있는 기법을 제안한다.

소프트웨어에서의 결합도는 모듈 또는 클래스간의 의존도를 의미한다. 처음으로 구조적인 개발을 위해 결합도를 도입한 Stevens et al의 결합도 정의를 따르면 “관계 강도의 측정은 한 모듈에서 다른 모듈로의 연결에 의해 설립된다(the measure of the strength of association established by a connection from one module to another)”와 같다. 따라서 강한 결합도를 가진 모듈 또는 클래스는 이해하거나 변경하기가 어려우며, 시스템의 정확도나 성능을 떨어뜨릴 수 있다[1][2]. 객체지향 시스템에서 결합도는 한 클래스와 다른 클래스의 연결로서 설명 될 수 있는데, 객체지향 개발 기법에서 클래스간의 관계를 형성하는 다양한 매커니즘이 존재하기 때문에 이를 분석하기 어렵다. 기존의 개발 방법과는 판이하게 객체지향 개발 기법에서는 클래스들 간의 관계가 매우 중요하며, 이 관계를 어떻게 형성하고 구현할 것인지 결정하는 문제는 시스템 전체의 성능이나 수행 속도, 복잡도 등을 결정하는 요소이기 때문에 매우 중요하다.

객체지향 방법론에서 절대적인 원칙으로 사용되는 높은 응집도와 낮은 결합도 원칙(High Cohesion, Low Coupling principle)은 소프트웨어의 유지보수성을 높이고 객체지향 패러다임의 장점을 최대로 사용할 수 있다(Larry Constantine, Edward Yourdon, 1970). 즉 객체지향 설계의 목표는 결합도를 낮추고 응집도를 높이는 것이다[3]. 소프트웨어의 품질을 최대한 높이고 유지보수성을 높이기 위해 최근에는 클래스가 아닌 인터페이스를 이용하여 구성하는 디자인 패턴이나 Mashup, SOA 등의 설계 기법을 많이 사용하고 있다. 그러나 이러한 방법들을 통해 소프트웨어 설계에 적용한다고 해도 가장 적합한 방법이

무엇인지, 어느 객체의 관계에 새로운 방법을 적용시킬 것인지를 결정하는데에는 어려움이 따른다.

본 논문에서는 객체지향 방법론에서 결합도 측정 메트릭을 이용하여 클래스들 사이의 관계를 측정하고 각 클래스들의 영향도 측정을 통해 소프트웨어 내부의 복잡도와 성능, 정확도를 도출해내는 일종의 프로세스를 제안한다. 또한 설계 단계에서 측정되는 정적 메트릭 측정 방법이 아닌 수행과 테스트 단계에서 측정되는 동적인 측정을 하기 위해 실제 소프트웨어의 영향도를 측정하기 위한 실험 집합을 생성해주는 Taguchi Method를 사용한다.

2. 관련 연구

2.1. SVM

SVM은 1998년 통계학자인 Vapnik에 의해 개발된 학습기법으로, 입력공간과 관련된 비선형문제를 고차원의 특징공간의 선형문제로 대응시켜 나타내기 때문에 수학적으로 분석하는 것이 수월하다[11]. 또한, SVM은 조정해야 할 파라미터의 수가 많지 않아 비교적 간단하게 학습에 영향을 미치는 요소들을 규명할 수 있다. 그리고 구조적 위험을 최소화함으로써 과대적합문제에서 벗어날 수 있으며, 볼록 함수를 최소화하는 학습을 진행하기 때문에 글로벌 최적해를 구할 수 있다는 점에서 인공신경망과 비교해 더 성능이 좋은 기계학습기법으로 주목 받고 있다.

SVM의 작동 원리는 크게 2가지로 설명될 수 있다. 첫 번째는 'maximum margin classification'의 원리이다. SVM은 기본적으로 서포트 벡터라 불리는 두 집단의 경계에 있는 점들로부터 가장 멀리 떨어져 있는 분류기(classifier)를 찾도록 설계되어 있다. 이를 통해, 두 집단을 가장 잘 나눌 수 있는 선형 분류기를 선형적으로 제약된 이차계획(QP) 문제(linearly constrained quadratic programming)로 정형화하여 찾을 수 있도록 설계되어 있다[4].

두 번째 SVM의 핵심 작동원리는 '고차원 공간으로의 데이터 위치이동(mapping data into higher dimensional space)'이다. 저차원 공간에서 선형 분류기로 분류되지 않는 문제들도 보다 높은 차원의 공간으로 사상시킬 경우, 선형 분류기로 더 잘 분류될 수 있다는 것이 SVM의 또 다른 핵심 작동원리이다[4].

2.2. 결합도(Coupling)

결합도는 모듈 또는 클래스들이 상호 의존하는 정도를 나타내는 것으로 모듈이나 클래스들 간의 결합도를 최소화시켜야 모듈의 독립성이 증가되며, 시스템 전체의 성능이 좋아지고 복잡도가 줄어든다. 그러나 두 개의 모듈이 서로 강하게 연결되어 상호간의 메시지 전송이 잦고, 주고받는 데이터들의 수가 많은 경우는 시스템 자체에 이해도가 떨어지며, 복잡도가 증가하고 객체지향 패러다임에서 적절하게 설계되었다고 할 수 없다[5].

결합도를 측정하는 방법은 지금까지 다양하게 연구되어 왔다. 각각의 결합도 측정 방법은 다양하게 소프트웨어의 결합도를 측정하며 측정 결과 또한 각 소프트웨어를 이해하는 방식대로 도출된다. 본 논문에서는 결합도 측정을 위한 메트릭을 간단하게 소개한다.

-MM(Method-Method interaction)

MM(Method-Method interaction)은 클래스 C에서 구현된 클래스 D의 정적 호출 메소드나 또는 같은 메소드에 포인터를 받는 것으로 정의한다. MM은 정의에서 분명하게 호출의 빈도를 계산하지 않고 클래스 메소드나 시그니처(문법)을 통해 계산되므로 정적 결합도 측정 방법에 해당한다. 이 MM의 측정 메트릭의 일반화 공식은 다음과 같다[6].

$$Metroc(c_i) = \sum_{c_j \in Relationship(c_i)} Interaction(c_i, c_j) \quad \cdots\cdots\cdots(1)$$

-IPC(Information Flow-based Coupling)

IPC(Information Flow-based Coupling)는 메소드를 호출하는 수를 카운트하여 클래스들의 가중치를 매개변수의 수에 따라 다르게 부과하는 정보흐름에 기반한 커플링 측정 방법이다. 클래스 사이의 정보흐름, 즉 메소드 호출의 빈도를 이용하여 측정하는 것이기도 하자만 객체 호출에서 클래스의 메소드로부터 계산되기도 한다[7].

-MPC(Message Passing Coupling)

MPC(Message Passing Coupling)은 다른 클래스에서 한 클래스의 메소드에서 호출하는 문장의 개수로 결합도를 측정하는 방법이다. 메시지 전송의 문장은 실제 객체의 메소드 호출의 실행 수를 반영하지 않는다[8].

-RFC(Response for Class)

RFC(Response for Class)는 클래스의 반응을 나타내는 원소를 포함하는 집합의 개수를 측정하는 방법이다. 반응 집합(Response set)은 모든 지역적인 메소드에서 다른 지역적인 메소드를 호출하는 클래스들로 이루어져 있다. 즉 RFC는 지역 메소드와 이 지역 메소드에서 호출하는 따른 지역 메소드들로 구성되어 있다[9].

-CBO(Coupling Between Objects)

Chidamber et.al.은 결합되어 있는 클래스의 개수로서 객체 사이의 결합도인 CBO(Coupling Between Object)를 정의하였다. 좀 더 추가적으로 정의하자면 "두 클래스는 한 클래스의 메소드를 사용하거나 또는 인스턴스 변수의 메서드가 다른 클래스에 의해 정의되면 결합된다."라고 할 수 있다. 이 측정 방법은 실행되는 동안의 호출의 개수를 계산하지 않기 때문에 동적인 결합도 측정 척도는 아니지만 호출된 메소드의 변수의 개수로 계산된다[8].

-DIT(Depth of Inheritance Tree)

DIT(Depth of Inheritance Tree)는 클래스의 상속관계의

계층적 구조에서의 위치를 측정하는 메트릭이다. DIT 메트릭이 클수록 클래스를 유지하는 것이 더욱 어려워 진다. 최상위 루트 클래스의 경우 DIT는 0이 된다[9].

-NOC(Number of Children)

NOC(Number of Children)는 현재 클래스에서 연관된 자식클래스를 측정할 수 있는 메트릭이다. DIT가 계층적 구조에서 클래스의 위치를 측정하는 메트릭이라면 DIT는 직접적으로 연관된 자식 클래스의 개수를 가지고 측정하는 방법이다[9].

-CP(Coupling Factor)

CP(Coupling Factor)는 결합도를 메시전달 또는 클래스 인스턴스간의 의미있는 연결로 의해 정의된다. 따라서 동적 결합도 측정에 이용되는 방법이다. CP는 클래스들의 상호작용에 가능한 다양한 방식의 정보 교환을 포함하는 결합도 측정 방식이다. CP는 다음 식2와 같이 정의되어 있다[10].

$$CP = \frac{\sum_{i=1}^{TC}\sum_{j=1}^{TC}[is\_client(C_i, C_j)}{TC^2 - TC}$$

$$is\_client(C_c, C_s) = \begin{cases} 1 iff C_c \Rightarrow C_s \wedge C_c \neq C_s \\ 0 \quad otherwise \end{cases} \quad \cdots\cdots\cdots\cdots (2)$$

2.3. Taguchi Method

Taguchi Method는 실제 실험의 견고한 설계를 통해 과정의 변화를 측정하고 그 측정 결과를 통해 변화를 감소할 수 있도록 하는 방법이다. Taguchi Method의 목적은 저렴한 비용으로 고품질의 결과를 생산하게 만드는 것에 있다. Taguchi Method는 Genichi Taguchi에 의해 개발되었으며, 서로 다른 매개 변수를 통해 프로세스가 얼마나 잘 작동하는지 성능의 특성을 평균과 분산에 미치는 영향에 대해 조사하기 위해 실험을 설계하는 방법을 제공한다. Taguchi Method의 실험 설계 배열을 설정하는 선택기는 아래의 [표 1] 과같다[12].

[표 1] Taguchi method orthogonal array Selector

Number of Levels	Number of Parameters (P)																												
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	L4	L4	L8	L8	L8	L8	L12	L12	L12	L12	L16	L16	L16	L16	L32	L32	L32	L32	L32	L32	L32	L32	L32	L32	L32	L32	L32	L32	L32	L32
3	L9	L9	L9	L18	L18	L18	L18	L27	L27	L27	L27	L27	L36	L36	L36	L36	L36	L36	L36	L36	L36	L36								
4	L'16	L'16	L'16	L'16	L'32	L'32	L'32	L'32	L'32																					
5	L25	L25	L25	L25	L25	L50	L50	L50	L50	L50	L50																			

3. 소프트웨어의 영향력 측정 및 분석

3.1. 소프트웨어 사용자 패턴 추출

본 연구는 소프트웨어의 결합도 측정 메트릭을 이용하여 소프트웨어의 수행중의 객체간의 상호작용과 영향도를 분석하여 실제 특정 객체의 파급력을 측정하는 측정 지표를 설정하고 이에 따른 분석을 위한 시스템을 설계 하였다.

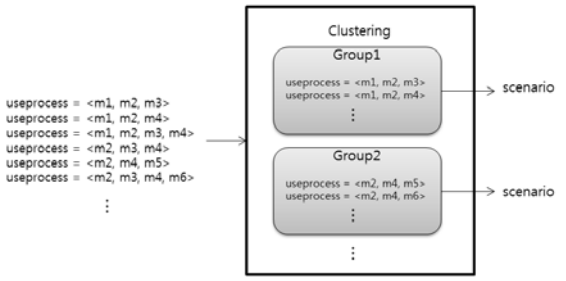
우선 첫 번째로, 소프트웨어 실제로 사용하는 사용자들의 패턴을 추출하여 동작하는 도중 다른 객체와의 관계를 이루고 호출하는 일련의 과정을 통해 소프트웨어의 실제 실행 상태에서의 객체들간의 상관관계를 분석한다. 이에 따라 소프트웨어의 정적 분석에서 사용되는 클래스 다이어그램의 결합도 분석과는 다르게 실제 사용자의 사용 사례를 통해 클래스 다이어그램에서는 도출해내기 어려운 실제 사용 영향도 그룹을 추출할 수 있게 된다.

설계 단계의 클래스 다이어그램은 실제 소프트웨어의 사용 빈도에 따라 중요도와 영향도가 달라질 가능성이 많기 때문에 실제 소프트웨어의 영향도를 측정하기 위해서는 소프트웨어를 실제로 이용하는 사용자들의 사용 패턴을 기반으로 소프트웨어가 동작하는 도중의 객체들의 관계와 호출 관계를 정의해야 한다.

소프트웨어를 사용하는 사용자의 패턴은 결국 사용자가 소프트웨어에 어떠한 입력 값을 정의하여 입력하고 그에 따른 결과를 받게 되는 일련의 과정을 의미한다. 따라서 사용자의 패턴을 추출하기 위해서는 사용자가 입력하는 데이터의 형태와 원하는 그에 따른 결과를 나타내는 프로세스를 거치게 된다. 사용자는 다양한 방식으로 소프트웨어에 접근하여 소프트웨어를 사용하게 되며, 소프트웨어 내의 각 모듈이 상호 작용하면서 결과를 나타낸다. 각 사용자가 소프트웨어를 사용할 경우 여러 가지 실행 흐름이 나타날 수 있다. 이러한 흐름을 다음과 같이 정의 할 수 있다.

$$M = m_a \rightarrow m_b$$
$$0 < i \leq |TM|$$
$$userprocess = \{x \mid x = F(M_i)\} \quad \cdots\cdots\cdots\cdots\cdots\cdots (3)$$

다양한 사용자의 useprocess가 모이게 되면 각 사용자의 사용 벡터를 통해 유사한 패턴을 나타내는 여러 사용 벡터를 통합하여 군집화 할 수 있게 된다. 사용자의 사용 벡터를 통합하여 SVM을 이용하여 유사한 패턴을 추출해 내고 나면 특정 군집 단위로 주요한 useprocess를 추출하여 동적 결합도를 측정할 시나리오로 표현한다. 전체 전체 흐름은 아래 (그림 1)과 같다.



(그림 1) 사용 흐름 클러스터링 과정

3.2. 시나리오 분석

이전 단계에서 소프트웨어의 사용 프로세스 벡터를 이

용하여 사용자의 사용 패턴에 따라 추출하였다. 그 결과를 실제 결합도 측정을 위한 시나리오의 형태로 변환하는 과정이 필요하다. 따라서 본 논문에서는 소프트웨어의 실제 수행 단계의 결합도를 측정하기 위해서는 사용자의 하나의 시나리오가 수행되는 동안 소프트웨어의 어떤 기능을 사용했는지, 또한 각 기능을 실행 가능하게 하기 위해 소프트웨어의 내부적인 실행 흐름을 정의한다. 사용자의 사용 시나리오의 형태는 아래의 [표 2]과 같다.

[표 2] 시나리오 형식

요소	설명
scenario	사용자 패턴 추출을 통해 결정된 테스트 시나리오
useprocess	사용자의 사용 흐름을 형식화 한 형태로, 클러스터링을 거친 후 각 군집들을 대표할 수 있는 사용 흐름을 의미한다.
Moduleprocess	시나리오의 흐름에 따라 필요한 모듈의 내부 동작 을 정의한 요소

시나리오는 사용자의 사용 흐름에 따라 클러스터 기법을 통해 군집화한 그룹을 특정할 수 있는 대표적인 사용 흐름을 선택한 것으로 각 그룹의 평균을 나타내는 사용 흐름이다. 이 사용흐름을 통해 각 그룹의 대표적인 시나리오를 추출할 수 있으며 이에 따라 각 내부적인 모듈의 동작을 서술한다. 두 번째로 useprocess는 시나리오를 표현하는 벡터 형태의 정보로서 실제 모듈이 동작하는 시간 흐름과 순서를 표현한다. 마지막 세 번째 요소인 Moduleprocess는 시나리오의 흐름에 맞게 모듈 내부의 동작을 상세히 서술한 내용으로 sequence diagram으로 표현한다. Sequence diagram은 실제 모듈 내부의 동작 방식을 클래스 단위의 메시지 전송까지 표현할 수 있는 방법으로 소프트웨어의 내부 실행 흐름을 가장 정확하게 표현하는 방식이다.

3.3. 수행 단계의 영향도 측정

실제 소프트웨어의 사용자 사용 시나리오에 따라 영향도를 측정하기 위해서는 수행 모듈의 각 파라미터에 따라 변화하는 결과와 그 결과에 따른 분석이 필요하다. 따라서 본 논문에서는 Taguchi Method의 실험 설계를 통해 실제 모각 모듈의 기본 실험 집합을 설정하고 그 수행 결과를 통해 결합도를 측정한다. Taguchi Method의 실험 설계는 다음과 같다.

이전 단계에서 추출된 사용자 사용 시나리오에 따라 모듈의 개수를 파라미터로 설정하고 각 모듈의 입력되는 변수를 Level로 설정하여 Taguchi Method의 실험 설계 메소드(Taguchi method orthogonal array Selector)를 결정한다.

[표 3] 사용자의 사용 흐름 시나리오

수행 모듈	입력 값	Level
Class1	문자열(String)	“abc”, “bbb”, “ccc”
Class2	상수(int)	50, 100, 150
Class3	상수(int)	15, 55, 75
Class4	객체(Class3)	Objec1, Object2

만약 사용자의 사용 흐름 시나리오가 위의 [표 3]과 같다면 Taguchi Method를 이용한 실험 설계는 표1에서 Taguchi Method를 이용한 실험 설계 선택기의 내용과 같이 L9를 사용한다. 입력값은 Taguchi Method에서 Level을 의미하며, 각 수행 모듈은 입력값이 고정되어 있지 않으므로 1~5개의 대표 값을 통해 실험을 수행한다.

Taguchi Method에 따른 실험 설계 결과는 아래의 [표 4]와 같다.

[표 4] Taguchi Method 실험 설계

Experiment	Class1	Class2	Class3	Class4
1	“abc”	50	15	Objec1
2	“abc”	100	55	Objec2
3	“abc”	150	75	Objec1
4	“bbb”	50	55	Objec2
5	“bbb”	100	75	Objec1
6	“bbb”	150	15	Objec2
7	“ccc”	50	75	Objec2
8	“ccc”	100	15	Objec2
9	“ccc”	150	55	Objec1

실험 결과는 수행되는 동안의 소프트웨어의 성능과 수행 시간을 의미하며, 해당 실험 결과에 따라 Taguchi Method의 영향도 측정 방법에 따라 가장 큰 영향력을 가진 모듈을 추출할 수 있다.

[표 5] 실험 측정 결과

Experiment	수행 시간(초)
1	18.1
2	19.54
3	12.3
4	11.9
5	19.3
6	20.21
7	15.97
8	24.3
9	22.5

만약 수행 결과가 [표 5]와 같이 측정되었다면, 그에 따른 영향도 측정은 아래의 계산과 같다.

$$\mathrm{Result}(Class1, Level1) = \frac{\sum Experiment(Class1, Level1)}{|Level1|}$$

$$\Delta = Max(Class1) - Min(Class1) = 20.92 - 16.64 = 4.28 \quad \cdots\cdots(4)$$

[표 6] 영향도 분석 결과

Level	Class1	Class2	Class3	Class4
1	16.64	15.32	20.87	18.5
2	17.13	21.04	17.98	18.384
3	20.92	18.33	15.85	
Δ	4.28	5.72	5.02	0.016
Rank	3	1	2	4

3.4. 결합도 분석

위의 예시에서 영향도 분석을 통해 Class2가 소프트웨어 사용자 사용 시나리오에서 가장 큰 영향도를 가진 것으로 분석되었다. 따라서 Class2와 관련된 클래스들 사이의 결합도를 분석하여 가장 큰 영향도를 가진 Class2와 어떤 관계를 가지고 있는지를 확인해야 한다. 따라서 마지막으로 클래스간의 결합도 측정을 위해 소개되었던 결합도 측정 메트릭인 CP와 MM를 결합한 형태인 식3을 사용하여 각 클래스간의 결합도를 측정한다. 결합도 측정을 한 후에 실제 영향도와 어떠한 차이가 있으며 실제 소프트웨어의 수행 도중 가장 큰 영향력을 가진 모듈의 영향도가 어디까지 파급력을 가지는지를 확인한다.

$$Metroc(c_i) = \sum_{c_j \in Relationship(c_i)} Interaction(c_i, c_j)$$

$$CP(c_j) = \frac{\sum_{i=1}^{TC} Metroc(c_i)}{TC^2 - TC} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (5)$$

[표 3]의 사용자 사용 프로세스를 통해 영향도를 측정한 결과 Class2의 영향도가 가장 높은것으로 나타났다. CP결합도 측정을 통해 메트릭을 구성했을 경우 아래의 [표 7] 같은 결과가 나왔다고 가정하자.

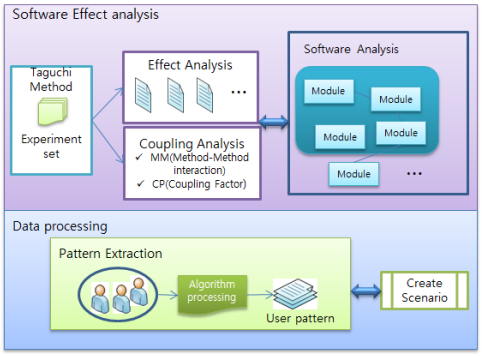
[표 7] 결합도 측정

	Class2
Class1 CP	$\frac{Metroc(Class2, Class1)}{TC^2 - TC} = 35\%$
Class1 CP	$\frac{Metroc(Class2, Class3)}{TC^2 - TC} = 25\%$
Class1 CP	$\frac{Metroc(Class2, Class4)}{TC^2 - TC} = 30\%$

영향도 분석 결과 소프트웨어의 동작 중 가장 큰 영향을 미치는 모듈이 Class2였다. 그 결과를 통해 Class2가 다른 모듈에 미치는 영향도의 실제적인 결과를 CP와 MM의 결합도 측정을 통해 표 7과 같이 표현 된다. 이에 따라 영향도를 분석한 후에 실제적으로 가장 큰 영향도를 가진 모듈이 다른 모듈과 어느정도 밀접한 관련성을 가지고 있는지를 확인할 수 있게 되었다.

3.5. 영향도 분석 시스템 아키텍처

지금까지의 소프트웨어 동작 영향도 분석 과정을 아키텍처로 표현하면 아래 (그림 2)와 같다.



(그림 2) 시스템 아키텍처

처음 사용자 그룹에서 소프트웨어의 사용 패턴을 추출한 후 그 사용 패턴을 기반으로 사용자의 사용 시나리오를 만들게 된다. 소프트웨어를 사용하는 실제 사용자의 사용 패턴을 통해 시나리오를 작성하게 되었으므로 소프트웨어의 동작 당시의 영향도와 결합도를 측정할 수 있게 된다. 또한 작성된 시나리오를 통해 사용 시나리오의 동작을 수행하는 모듈들의 파라미터와 그 결과를 측정 할 수 있는 실험 집합을 만들 수 있는 Taguchi Method로 실험 집합을 생성한다. 생성된 실험 집합을 가지고 소프트웨어의 수행 결과를 통해 영향도 측정과 결합도 측정을 수행하고 분석한다. 이렇게 측정된 영향도와 결합도의 결과를 통해 소프트웨어에서 특정 모듈이 전체 소프트웨어의 동작에서 어느 정도까지 영향도를 미치고 파급력을 가지게 되는지를 분석할 수 있게 된다. 본 논문에서는 하나의 시나리오를 가정하여 영향도와 결합도 측정을 수행하였지만 실제 사용자의 소프트웨어의 동작을 통해 나올 수 있는 여러 개의 사용 패턴을 통해 측정을 하고, 그 측정 결과를 통합하여 분석한다면 전체 소프트웨어의 분석이 가능해 진다.

5. 결론

지금까지 소프트웨어의 실제 사용자의 사용 패턴을 추출하고 그 패턴을 통해 소프트웨어의 각 모듈의 상관 관계와 영향도, 결합도를 분석하여 실제 소프트웨어의 동작 도중 어떤 모듈이 가장 큰 파급력을 가지고 있는지를 확인하고 분석할 수 있는 시스템을 제안하였다. 사용자의 사용 사례를 통해 패턴을 추출하고 이를 통해 실험 데이터를 만들어 실제 소프트웨어의 내부 모듈간의 관계를 측정할 수 있는 시스템을 제안함으로써 정적 소프트웨어의 측정 단계에서 발견하지 못한 소프트웨어의 결함을 추출하거나 소프트웨어의 재사용성, 유지보수성을 증대시킬 수 있게 된다.

향후 연구로는 실제 소프트웨어의 시스템이 본 제안 시스템을 적용시킴으로써 어느정도의 효용성을 나타낼 수 있는지를 확인하고, 동적인 측면에서의 소프트웨어의 영향도를 측정 및 실험함으로써 기존의 정적 결합도 측정 메트릭과의 비교 분석을 할 예정이다.

참고문헌

[1] R.W. Selby and V.R. Basili, “Analyzing Error-Prone Systems Structure,” IEEE Trans. Software Eng., vol. 17, no. 2, pp. 141-152,1991.

[2] P.A. Troy and S.H. Zweben, “Measuring the Quality of Structured Designs,” J. Systems and Software, vol. 2, pp. 113-120, 1981.

[3] N. Renton and S. Pfleeger, Software Metrics: A Figorous & Practical Apptoach, PWS Publishing Company, 1997

[4] Platt, and B. Scholkopf, “Support vector machines”, IEEE Intelligent System, Vol. 13, No. 4, 1998, pp. 18-28.

[5] Roger S. Pressman, “Software Engineerignn A Practitioners Approach, Fourth Edition” McGrawHill, 1997

[6] Briand, L., P.Devanbu, and W.Melo, “An Investigation into Coupling Measures for C++,” Proc. Of the 19th International Conference on Software Engineering, ICSE'97, Boston, May 1997, pp412-421

[7] Lee, Y., B. Liang, and S. Wu, “Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow,” Proc. Of International Conference on Software Quality, Maribor, Slovenia, 1997

[7] Li, W., and S. Henry, “Object Oriented Metrics that predict Maintainability” Journal of Systems and Software, Vol23, No. 2, 1993, pp111-122

[8] Chidamber, S., and C.Kemerer, “A Metrics Suite for Object Oriented Design”, IEEE Transactions on Software Engineering, Vol 20, No 6, June 1994, pp476-493

[9] Rombach, H. Dieter, “Design Measurement: Some Lessons Learned,” IEEE Software, March 1990, pp.17-25

[10] Ruchika Malhotra, "Empirical Study of Object-Oriented Metrics", Published by ETH Zurich, Chair of Software Engineering, 2006

[11] Hearst, M.A., S.T. Dumais, E. Osman, J. Vapnik, V., Statistical Learning Theory, Wiley, 1998.

[12] Stephanie Fraley, Mike Oom, Ben Terrien, John Zalewski, “Design of experiments via taguchi methods: orthogonal arrays”, The Michigan Chemical Process Dynamics and Controls Open Text Book, 2006.