로지스틱 분석을 이용한 메소드 위치 결정 방법

정 영 애[†] · 박 용 범[††]

요 약

소프트웨어의 요구사항 변경은 소프트웨어의 생명주기 전반에 걸쳐 발생한다. 이러한 변경은 소프트웨어의 수정을 요구하며, 소프트웨어 수정시 소프트웨어의 품질과 안정성을 향상시키는 것은 중요한 문제이다. 리팩토링은 소프트웨어의 품질과 안정성을 보장하면서 소프트웨어를 수정하는 기술이다. 따라서 리팩토링의 자동화에 대하여 다양한 연구가 이루어지고 있다. 본 논문에서는 무브 메소드(Move Method)의 적용여부를 결정지을 수 있는 세 가지 요인을 정의하였다. 정의된 요인에 의해 데이터를 샘플 프로그램에서 추출하였고, 추출된 데이터에 이진 로지스틱 회귀분석을 적용하였다. 이진 로지스틱 회귀분석을 통하여 얻은 무브 메소드 적용 여부에 대한 추측결과는 숙련된 프로그래머들의 수동분석 결과와 상당부분 일치하였다. 더불어, 각 요인들은 프로그램 내에서 메소드의 위치를 결정하는데 중요하게 작용하며, 메소드의 최적 위치를 결정짓는 기준으로써 사용될 수 있음을 밝혔다.

키워드 : 리팩토링, 자동화, 로지스틱 분석, 메소드 위치 결정

An Approach to decide the location of a method using the logistic analysis

Young A. Jung[†] · Young B. Park[††]

ABSTRACT

There are many changes in the software requirements during the whole software life cycle. These changes require modification of the software, and it is important to keep software quality and stability while we are modifying the software. Refactoring is one of the technology to keep software quality and stability during the software modification; there are many researches related to automatic refactoring. In this paper, we propose three factors for Move Method which is one of the refactoring technique. We applied binomial logistic analysis to data which were extracted from sample program by each factor. The result of this process was very close to the result of manual analysis by program experts. Furthermore, we found that these factors have major roll to determine position of a method, and these factors can be used as a basis of finding optimal position of a method.

Key Words : Refactoring, Automation, Logistic Analysis, Decision of Method Location

1. 서 론

일반적으로 소프트웨어는 생명주기 전반에 걸쳐 요구사항의 변경을 수용하기 위해 변경된다. 이런 종류의 소프트웨어 유지보수 활동은 프로그램 모듈의 검증과 여러 변경사항에 대한 테스트가 소프트웨어를 사용하지 않을 때까지 지속적으로 이루어져야 한다는 점에서 상당한 비용이 요구된다[8].

소프트웨어 개발자들은 처음부터 완벽한 소프트웨어를 개발하고자 한다. 하지만 처음부터 완벽한 소프트웨어를 만드는 것은 불가능하다. 대부분의 경우 코드가 작성된 후 더 자주 수정되기 때문에 가독성(readability)과 유지보수성(maintainability)이 요구되어 왔다. 또한 최근에는 요구사항의 변경과 관련된 유지보수를 좀 더 체계적이고 안전하게 수행하기 위하여 리팩토링의 사용이 증가하고 있다.

리팩토링의 수행은 대부분 리팩토링 전문가들의 경험에 의한 수동 분석(manual analysis)을 통하여 이루어지고 있다. 소프트웨어에 리팩토링을 적용하기 위한 체계적인 기준이 제공되지 못하기 때문이다[8]. 현재 프로그래머가 전문가로부터 얻은 간접경험이나 자신의 필드경험을 바탕으로 새로운 리팩토링 작업에서 올바른 리팩토링을 수행하는 방법을 도입하여 리팩토링 작업의 상당 부분을 자동화하려는 시도가 활발하다[9, 10]. 리팩토링을 자동화하기 위해 필요한 객관적인 판단기준과 방법도 다양하게 제시되어 왔다[3, 4, 8-10]. 더불어 자동화를 위한 각 기법별 요인분석 또한 중요하다. 본 논문에서 프로그램 내의 메소드의 위치를 결정할 수 있는 요인을 분석하여, 리팩토링 기법 중 메소드의 위치와 관련하여 빈번히 사용되는 무브 메소드 기법의 적용요인

※ 이 연구는 2004학년도 단국대학교 대학연구비의 지원에 의해 수행되었음.
† 정 회 원 : 단국대학교 대학원 전자계산학과
†† 종신회원 : 단국대학교 전자컴퓨터학부 부교수
논문접수 : 2005년 9월 21일, 심사완료 : 2005년 12월 5일

을 다음과 같이 세 가지로 정의하였다.

적용요인1에서는 대상 메소드가 참조하는 소스클래스와 타겟클래스의 필드 개수의 차이를, 적용요인2에서는 일반 메소드에 대한 메소드 참조횟수의 차를 비교하였고, 적용요인3에서는 소스클래스와 타켓 클래스 안의 일반 메소드들이 대상 메소드를 호출하는 빈도수의 차를 비교하였다.

이 세 가지 요인은 파울러의 나쁜 냄새(bad smell)와 서로에게 속한 것은 같은 곳에 둔다(put together what belong together)는 기본적인 개념에 초점을 맞추어 구하였다. 객체 지향 프로그래밍에서 메소드는 멤버변수(attribute)와 더불어 기본적인 요소이다. 그것이 어떤 위치에 있는가는 프로그램의 안정성에 큰 영향을 미친다. 본 논문에서는 이 세 가지 요인이 각 객체가 포함하는 메소드의 최적의 위치를 결정짓는 요소로써 적절한가를 분석하였다.

정의된 세 가지 요인별 사례분석은 이진 로지스틱 회귀분석을 사용하였다. 무브 메소드의 적용여부에 대한 분석결과를 숙련된 프로그래머들의 수동분석의 결과와 비교하여 상당부분 일치함을 보였다. 이 결과로부터 각 요인이 여러 객체들과 메소드 간의 결합관계를 결정지을 수 있는 요인으로써 작용함을 알게 되었다.

본 논문의 구성은 다음과 같다. 2장에서는 리팩토링과 자동화 동향, 사례 분석에 사용된 리팩토링 기법인 무브 메소드(Move Method)와 로지스틱 회귀분석에 대하여 설명한다. 3장에서는 객체 안에 있는 메소드의 위치를 결정하기 위하여 사용하게 될 무브메소드 적용요인을 추출하고 정의하였으며 4장에서는 이렇게 정의된 요인들을 바탕으로 이진 로지스틱 회귀분석을 이용하여 얻은 추측 결과와 기존 프로그램 전문가들의 수동분석과 비교하였다.

2. 기존 연구

2.1 리팩토링

리팩토링은 프로그램의 행위(behavior)를 보전하면서 프로그램의 가독성, 구조, 성능, 유지보수성, 추상성 등을 향상시키는 좋은 방법이다[1]. 소프트웨어 개발자는 리팩토링을 사용하여 프로그램을 이해하기 쉽고 변화를 포용할 수 있도록 프로그램을 단계적으로 개선할 수 있다. 리팩토링은 프로그램의 가독성을 높일 뿐만 아니라 개발 속도를 높일 수 있도록 효율적이고 통제된 방법을 제공하여 소프트웨어의 가치를 높인다는 장점을 가진다[1]. 따라서 많은 개발자들은 리팩토링 자동화를 위하여 지속적인 연구를 하였고, 그 결과도 매우 다양하다.

리팩토링은 1990년 초에 개발되었고[11-13], 그 후 소프트웨어 개발의 주된 분야로 대두 되었다[7]. 초기의 리팩토링은 대부분 수동분석에 의존하였다. 리팩토링 자동화는 리팩토링이 필요한 후보영역을 정의하는 분야와 개발자가 정의한 후보영역에 리팩토링을 자동으로 적용하는 분야로 양분화 되었다[14].

리팩토링의 후보 영역을 정의 하는 분야의 대표로는 프로그램 불변점(program invariants)을 이용한 접근방법이 있다. 이 영역에서는 특정 리팩토링 기법에 대한 불변점 패턴 매칭(invariants pattern matching)을 사용하여 리팩토링이 필요한 부분을 찾아 나가는 방법을 사용한다[6, 14].

또 다른 연구로 요구사항의 변화에 대하여 자바 프로그램의 후보영역(candidate spot)을 원하는 디자인 패턴 구조로 변환하는 방법이 제안되었다. 이 연구에서는 후보영역을 원하는 디자인패턴으로 자동변환하기 위하여 높은 유연성을 가지는 디자인 패턴을 기초로 하는 추론(inference) 규칙이 사용되었다[9].

개발자 정의 후보영역에 자동으로 리팩토링을 적용하는 영역에서는, 프로그램 슬라이싱(Program slicing)을 사용하여 객체지향 프로그램의 메소드를 자동으로 리팩토링하는 방식[5], 가중치를 가지는 종속 그래프(weighted dependence graph)를 이용하여 객체지향 프레임워크에서 메소드를 자동으로 리팩토링 하는 메커니즘[4]등이 제안되었다.

더불어 결합도 매트릭스(cohesion matrix)를 이용한 거리를 측정하여 시각화(visualisation)하는 툴을 개발하여 리팩토링을 돕는 연구도 제안되었다[3].

본 논문에서는 리팩토링의 기법 중 하나인 무브 메소드 기법 요인을 추출하여 그 요인을 메소드의 위치를 결정 짓기 위한 요인으로 사용할 수 있는가에 대한 연구를 실시하였다.

2.2 로지스틱 회귀분석

로지스틱 회귀분석이란 독립변수의 양적인 변수를 이용해서 종속변수인 이변량적 변수들간의 인과관계를 추정하는 기법이며, 한 개의 종속 변수와 여러 개의 독립변수간의 상호관련성에 대해 분석하려 할 때 가장 널리 사용되는 통계기법이다.

종속변수가 두 가지 값만 취하는 질적인 이변수이고 독립변수가 k개인 경우에 선형회귀모형은 (식 1)과 같이 정의된다.

$$\text{logit}(P) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k \quad \text{(식 1)}$$

로지스틱 회귀분석 모델은 추후확률(posteriori probability)의 로지스틱 변환(transformation)이 독립 변수간이 선형함수라는 가정을 한다. 여기서 로지스틱 변환이란 다음 (식 2)와 같이 정의된다.

$$\text{logit}(P) = \ln\left(\frac{p}{1-p}\right) \quad \text{(식 2)}$$

따라서 로지스틱 형태의 확률 값은 (식 3)과 같이 표기될 수 있다.

$$\text{logit}(P) = \log\ odds = \ln\left(\frac{P}{1-P}\right) \quad \text{(식 3)}$$

이때 각 입력변수가 분류결정을 내리는데 미치는 영향의 정도를 Odd Ratio라 하며 이는 분석에 중요한 영향력을 행

사하게 된다. 각 x_i에 대한 odds 변환은 확률이 양수만을 취하므로 (식 4)와 같은 비례성질을 가진다.

$$e^{\beta_i} = \frac{e^{\beta_0+\beta_1 x_1+\cdots+\beta_i(x_i+1)+\cdots+\beta_k x_k}}{e^{\beta_0+\beta_1 x_1+\cdots+\beta_i x_i+\cdots+\beta_k x_k}} \quad \text{(식 4)}$$

일반 회귀분석은 종속변수와 독립변수가 연속적이므로 종속변수가 불연속적일 때는 일반 회귀 분석이 적합하지 않으며, 판별분석이나 로지스틱 회귀분석을 이용하여야 한다.

판별분석은 매트릭 척도에 의해 측정된 독립변수들을 사용하여 각 집단차를 최대로 하는 성형결합 함수를 추출한다. 그 함수를 통하여 추가적인 대상에 대한 집단 분류를 예측할 수 있다. 따라서 판별분석은 독립변수들이 연속변수일 것을 가정하지만, 로지스틱 회귀분석은 이산변수에 대한 비선형적 관계를 규명하게 된다. 또한 종속변수의 척도가 연속형이 아니라 범주형으로 측정되기 때문에 객체 매소드의 위치를 분석하는 데는 로지스틱 회귀분석을 이용하는 것이 적합하다[2].

따라서 본 논문에서는 로지스틱 회귀분석 방법을 선택하였다. 무브 메소드의 적용요인으로 정의된 세 가지 요인을 로지스틱 회귀분석의 독립변수로서 사용하여, 각 적용요인들과 무브 메소드의 적용여부에 대한 인과관계를 추정하도록 한다.

3. 로지스틱 기법을 위한 메소드 이동 결정 요인 선정

무브 메소드를 적용하는데 있어 다양한 상황을 판단하여 메소드를 옮길 것인지에 대한 확신을 가지는 것은 쉬운 일이 아니다. 본 연구에서는 수동적 분석에 의해 리팩토링이 수행될 때 보편적으로 사용되는 요인들을 유도하였다. 무브 메소드의 적용요인 즉 메소드의 위치를 결정짓는 요인의 유도를 위해 다음을 정의하였다.

- 소스클래스(source class) : 무브 메소드를 적용할 메소드가 포함되어 있는 클래스
- 타겟클래스(target class) : 무브 메소드를 적용할 메소드가 이동하게 될 클래스
- 필드(field) : 무브 메소드를 적용할 메소드가 참조하는 소스클래스의 필드 (타켓 클래스의 필드를 참조하기 위하여 get/set 메소드를 사용하는 것은 타켓 클래스의 필드를 참조하는 것으로 간주함)
- 대상 메소드 : 무브메소드기법을 적용할 메소드
- 일반 메소드(general method) : 타켓 클래스의 필드를 참조하기 위한 get/set 메소드를 제외한 모든 메소드

메소드의 위치를 결정할 수 있는 세 가지 요인은 파울러가 정리한 무브 메소드 기법에 기초하였다.

파울러는 "Move Method를 메소드가 자신이 정의된 클래스보다 다른 클래스의 기능을 더 많이 사용하고 있다면, 이 메소드를 가장 많이 사용하고 있는 클래스에 비슷한 몸체를 가진 새로운 메소드를 만들어라. 그리고 이전 메소드는 간단한 위임으로 바꾸거나 완전히 제거하라."고 정의하였다[7].

본 논문에서는 대상 메소드가 타겟클래스의 필드를 소스클래스의 필드보다 많이 사용하거나, 공동으로 일하는 부분이 너무 많아서 단단히 결합되어 있는 경우를 고려하였다. 이런 경우 메소드를 옮김으로써 클래스의 역할을 좀 더 명확하게 구현할 수 있다.

무브 메소드 적용요인1에서는 대상 메소드가 참조하는 소스클래스의 필드 개수와 get/set 메소드를 사용하여 참조하는 타겟클래스의 필드 개수의 차이를 비교한다. 소스클래스와 타겟클래스의 상반된 경우를 생각해 보면 get/set 메소드는 참조 클래스의 필드개수라 할 수 있다. 결과적으로, 적용요인1은 대상 메소드가 참조하는 필드의 개수에 대한 비교가 되며, 대상 메소드는 참조하는 필드의 개수가 많은 클래스로 이동한다.

무브 메소드 적용요인2는 일반 메소드에 대한 메소드 참조횟수의 차를 비교한다. 대상 메소드에서 소스클래스 내부에 있는 메소드보다 타겟클래스 내부에 있는 메소드에 대한 호출이 많다면, 무브 메소드를 적용한다.

무브 메소드 적용요인3은 소스클래스와 타켓 클래스 안의 일반 메소드들이 대상 메소드를 호출하는 빈도수의 차를 비교한다. 타겟클래스에서 호출하는 횟수가 많은 경우 타겟클래스와 강하게 결합하여 있으므로 무브 메소드 기법을 적용한다.

각 샘플 프로그램에 대하여 각 요인별로 얻어진 자료는 로지스틱 회귀분석의 독립요인으로 적용된다.

4. 실험방법 및 고찰

실험을 위하여 무브 메소드 리팩토링을 적용할 수 있는 프로그램을 선별하여 실험에 이용하였다. 각 프로그램별로 각 적용요인에 해당되어지는 개별적인 데이터를 구하였다. 각 적용요인에 의해 얻어진 데이터는 <표 1>과 같다. 소스클래스와 타겟클래스의 숫자는 각 적용요인에 따라 참조하는 필드의 개수나 참조횟수를 나타낸다.<표 1>에서 수동분석결과는 실험에 사용된 프로그램을 한 사람의 프로그래머가 분석한 결과이다.

<표 1>에서 구한 데이터를 로지스틱 회귀분석의 독립변수로 적용하기 위하여 소스클래스와 타겟클래스 데이터의 차를 구하였다.

Factor1 = 적용요인1의 소스클래스 데이터
- 적용요인1의 타겟클래스 데이터

Factor2 = 적용요인2의 소스클래스 데이터
- 적용요인2의 타겟클래스 데이터

Factor3 = 적용요인3의 소스클래스 데이터
- 적용요인3의 타겟클래스 데이터

〈표 1〉 정의된 적용요인에 의해 얻어진 데이터

프로그램 번호	적용요인1		적용요인2		적용요인3		수동분석 결 과
	소스 클래스	타겟 클래스	소스 클래스	타겟 클래스	소스 클래스	타겟 클래스	
1	0	0	0	1	0	1	1
2	1	0	1	1	0	1	0
3	0	2	0	2	1	0	1
4	0	0	1	0	0	0	0
5	0	3	1	0	1	0	1
6	2	0	0	0	1	0	1
7	2	0	0	0	1	0	1
8	2	0	0	0	1	0	1
9	2	0	0	0	1	0	1
10	1	2	0	0	1	0	1
11	2	0	0	0	1	0	1

〈표 2〉 이진 로지스틱 회귀분석에 실제로 적용될 데이터

프로그램 번 호	Factor1	Factor2	Factor3	수동분석 결 과
1	0	-1	-1	1
2	1	0	-1	0
3	0	-2	1	1
4	0	1	0	0
5	0	1	1	1
6	1	0	1	1
7	1	0	1	1
8	1	0	1	1
9	1	0	1	1
10	0	0	1	1
11	1	0	1	1

<표 2>는 위의 식을 이용하여 구한 데이터를 이용하여 로지스틱 회귀분석을 실시하기 위한 데이터이다.

이 과정에서 Factor 1~3을 소스클래스와 타겟클래스간의 대소를 비교하여 크다 또는 작다의 범주를 가지는 데이터에 대한 분석도 시도하였고 그 결과는 <표 2>에 의한 분석결과와 같았다.

각 적용요인의 조합에 따라 그 결과가 어떻게 달라지는지를 파악하기 위하여, 세가지 적용요인을 모두 적용한 경우, 세 가지 적용요인 중 두 가지씩 짝을 지은 3가지의 경우, 개별적인 경우 3가지에 대하여 분석을 수행하였다.

<표 3>은 세가지 요인을 모두 적용하였을 때의 결과이다.

<표 4>는 두 개의 요인을 적용한 세 가지 경우에 대한 로지스틱 분석결과이다. <표 4>에서 보는 바와 같이 세 가지 요인을 모두 적용할 경우와는 상이한 결과이다. 적용요인 2,3을 적용한 경우에는 예상그룹에 대한 예측이 수동분석결과와 100% 일치하였으나, 나머지의 경우에는 82%만이 일치하였다.

<표 5>는 적용요인들을 개별적으로 적용한 세 가지 경우에 대한 로지스틱 분석결과이다.

〈표 3〉 세 개의 요인을 모두 적용한 로지스틱 분석결과

프로그램 번 호	수동분석 결 과	Factor 1,2,3	
		예상확률 (predicted probability)	예상그룹 (predicted group)
1	1	0.9999999966	1
2	0	0.0000000000	0
3	1	1.0000000000	1
4	0	0.0000000104	0
5	1	0.9999999894	1
6	1	0.9999999997	1
7	1	0.9999999997	1
8	1	0.9999999997	1
9	1	0.9999999997	1
10	1	1.0000000000	1
11	1	0.9999999997	1
	일치확률		100%

〈표 4〉 두 개의 요인을 적용한 세 가지 경우에 대한 로지스틱 분석결과

수동분석결과	Factor 1,3		Fcator 1,2		Factor 2,3	
	예상확률	예상그룹	예상확률	예상그룹	예상확률	예상그룹
1	0.3397018856	0	1.0000000000	1	0.9999999904	1
0	0.2706739793	0	0.8333333333	1	0.0000000071	0
1	0.9603499280	1	1.0000000000	1	1.0000000000	1
0	0.7792482632	1	0.5000000000	1	0.0000000061	0
1	0.9603499280	1	0.5000000000	1	0.9999999869	1
1	0.9458651665	1	0.8333333333	1	1.0000000000	1
1	0.9458651665	1	0.8333333333	1	1.0000000000	1
1	0.9458651665	1	0.8333333333	1	1.0000000000	1
1	0.9458651665	1	0.8333333333	1	1.0000000000	1
1	0.9603499280	1	0.9999999941	1	1.0000000000	1
1	0.9458651665	1	0.8333333333	1	1.0000000000	1
일치확률		82%		82%		100%

〈표 5〉 개별적 요인으로 적용한 세 가지 로지스틱 분석 결과

수동분석결과	Factor 1		Factor 2		Factor 3	
	예상확률	예상그룹	예상확률	예상그룹	예상확률	예상그룹
1	0.8000000000	1	0.9775400000	1	0.3120000000	0
0	0.8333333321	1	0.8650000000	1	0.3120000000	0
1	0.8000000000	1	0.9966300000	1	0.9530000000	1
0	0.8000000000	1	0.4854000000	0	0.7520000000	1
1	0.8000000000	1	0.4854000000	0	0.9530000000	1
1	0.8333333321	1	0.8650000000	1	0.9530000000	1
1	0.8333333321	1	0.8650000000	1	0.9530000000	1
1	0.8333333321	1	0.8650000000	1	0.9530000000	1
1	0.8333333321	1	0.8650000000	1	0.9530000000	1
1	0.8000000000	1	0.8650000000	1	0.9530000000	1
1	0.8333333321	1	0.8650000000	1	0.9530000000	1
일치확률		82%		82%		82%

<표 5>에서 보는 바와 같이 개별적으로 적용한 경우에는 모두 82%만이 일치하였다.

이는 세 가지 적용요인들을 어떻게 조합하여 적용하느냐에 따라 다른 결과가 나올 수 있다는 사실을 확인해주는 결과이다.

위의 표에서 본 것과 같이 로지스틱 분석결과와 수동분석결과는 82% 이상 일치하였다. 또한 세 가지 적용요인을 모두 적용한 경우와 적용요인 2,3을 적용한 경우에는 100%의 일치율을 보였으나 그 외의 경우에는 82%의 일치율를 보였다. 이 결과로 미루어 적용요인 1은 적용요인 2나 3에 비하여 상대적으로 더 중요하게 작용함을 알 수 있다. 이는 메소드의 위치를 결정짓는데 대상메소드가 참조하는 필드의 개수보다는 대상메소드가 메소드를 호출하거나 호출되는 횟수가 더 중요하다는 것을 의미한다. 또한, 로지스틱 회귀분석에서 독립변수로 사용되는 적용요인의 수가 적은 경우보다는 많은 경우에 더 좋은 결과를 보여주고 있다.

자동화에 대한 연구는 대부분 디자인 패턴, 블록 슬라이싱, 가중치 그래프 등과 같은 컴퓨터 기반연구에 기초한다[4-6, 9, 14]. 그러나, 본 논문에서는 리팩토링 기반연구를 통하여 기본 요인을 찾아 메소드의 위치를 결정하는데 사용하기 위하여 그 요인들이 충분히 적용될 수 있는 요인이가에 대한 검사를 이진 로지스틱 회귀분석을 사용하였다. 본 논문과 같이 기존의 리팩토링의 연구에 기초하여 요인에 대한 추출과 이진 로지스틱 회귀분석을 통한 검증만 제대로 이루어진다면, 추출된 요인들은 프로그램에서 멤버 변수나 멤버 메소드 등의 위치를 결정할 수 있는 기준으로 사용할 수 있게 된다. 본 논문은 메소드의 위치결정을 위한 요인 분석에 한 개의 종속 변수와 여러 개의 독립변수간의 상호관련성을 분석하기 위해 통계기법을 적용함으로써 다양한 경우의 실험을 거쳐 일반적인 요인으로 사용할 수 있는 척도를 제공한다는 장점이 있다.

5. 결 론

리팩토링은 소프트웨어 시스템을 안정적으로 변환하는 방법으로 인식되어 왔다. 또한, 수동분석에 의존하던 리팩토링에서 다양한 방법으로 자동화에 대한 연구가 지속되고 있다.

본 논문에서는 리팩토링을 자동화하기 위한 연구를 기반으로, 리팩토링 기법 중 하나인 무브 메소드의 적용여부를 판단할 수 있는 세 가지 적용요인을 정의하였다. 이렇게 정의된 적용요인들은 로지스틱 회귀분석의 독립변수들로 사용되어 분석되었고, 그 분석결과를 통해 분석에 사용되는 적용요인의 조합에 따라 예측확률이 달라질 수 있음을 알 수 있었다. 또한 분석에 사용되는 적용요인이 많아질수록 정확한 결과를 예측할 수 있었다. 연구 결과 매우 흥미로운 사실은 무브 메소드 기법에서 필드의 개수보다는 메소드 참조 횟수가 무브 메소드 기법에 영향을 준다는 것이었다.

무브 메소드를 적용할 것인가 안할 것인가를 자동으로 결정하는 것은 매우 중요한 일이지만 본 연구는 자동 적용 결정보다는 메소드들을 대상으로 최적의 위치를 찾아줌으로써 소프트웨어의 안정성을 높이는데 그 목적이 있다. 본 논문에서 제시한 방법을 통하여 무브 메소드뿐만 아니라, 멤버 변수 옮기기(Move Attribute), 클래스 추출하기(Extract Class)와 클래스 합치기(Inline Class)등에도 적용할 수 있을 것으로 여겨진다. 다양한 리팩토링 기법에 대한 규칙을 유도하여 그것에 대한 분석 및 검증 과정을 거친다면, 소프트웨어의 멤버 변수들과 메소드의 최적의 위치를 결정할 수 있게 될 것이다.

참 고 문 헌

[1] 박지훈, '자바 디자인 패턴과 리팩토링', 한빛미디어㈜, 2003

[2] 성웅현, '응용 로지스틱 회귀분석', 도서출판 탐진, 2001.

[3] F. Simon, F. Steinbrükner, and C. Lewere- ntz, "Metrics based refactoring", in Proc. European Conf. Software Maintenance and Reengineering IEEE, pp.30-38, Computer Society, 2001

[4] Katsuhisa Maruyama, Ken-ichi Shima, "Automatic Method refactoring using weighted dependence graphs", Proceedings of the 21st international conference on Software engineering, 1999.

[5] Katsuhisa Maruyama, "Automated Method-extraction refactoring by using block-based slicing", Proceedings of the 2001 symposium on Software reusability: putting software reuse in context, Vol.26, pp.236-245, 2001.

[6] Kuyul Noh, Changki Kim, Jonggul Park and Jaeha Song, "The Evaluation of Daikon : utilization of Daikon in the POI Data Inspection System", 4WD Team Master of Software Engineering Program School of Computer Science, Carnegie Mellon University, 2002.

[7] Martin Fowler, etc, 'Refactoring Improving the Design of Existing Code', Addison Wesley, 1999.

[8] Sang-Uk Jeon, "An Approach to Automa- tically Identifying Design Structure for Applying Design Pattern", M.S. thesis of KAIST, 2003.

[9] Sang-Uk Jeon, Joon-Sang Lee, and Doo-Hwan Bae, "An Automated Refactoring Approach To Design Pattern-Based Program Transformations in Java Programs", In Proceedings of 9th Asia-Pacific Software Engineering Conference, Gold Coast in Austrailia, 2002.

[10] Stefan Roock , Andreas Havenstein, "Refactoring Tags for automatic refactoring of framework dependent applications", XP2002, 2002.

[11] William G. Griswold, 'Program Restruct- uring as an Aid to Software Maintenance PhD Thesis', Dept. of Computer Science & Engineering, University of Washington, 1991.

[12] William F. Opdyke and Ralph E.Johnson, Refactoring : An aid in designing application Frameworks and evolving

object-oriented systems, In Proceedings of SOOPPA '90 : Symposium on Object-Oriented Programming Emphasizing Practical Applications. Sep., 1990.

[13] W.F. Opdyke, 'Refactoring object-oriented frameworks Ph.D.thesis', Computer Sciences Department, University of Illinois at Ur bana-Champaign, 1992.

[14] Yoshio Kataoka, Michael D. Ernst, William G. Griswold, David Notkin, "Automated support for program refactoring using Invariants", Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01), pp.736-743, 2001

정 영 애

e-mail : yajung@dankook.ac.kr

1995년 호서대학교 전자계산학과(학사)

2000년 호서대학교 대학원 전자계산학과 (이학석사)

2003년~단국대학교 대학원 전자계산학과 박사수료

관심분야 : 소프트웨어 공학(소프트웨어 메트릭스, 프로젝트 관리, 소프트웨어 품질평가), 패턴인식

박 용 범

e-mail : ybpark@dankook.ac.kr

1985년 서강대학교 전자계산학과(학사)

1987년 N.Y.Polytechnic Univ. 대학원 전자계산학과(석사)

1991년 N.Y.Polytechnic Univ. 대학원 전자계산학과(박사)

1993년~현재 단국대학교 전자계산학과 부교수

관심분야 : Information Architecture, 패턴인식, 분산에이전트