메소드의 매개변수 리스트의 간소화를 위한 리팩토링 방안†

(Removing Long Parameter List Using Semantic Matrix)

함동화‡ (Dong Hwa Ham) 이준하‡ (Jun Ha Lee) 박수진§ (Soo Jin Park) 박수용¶ (Soo Young Park)

요 약 소프트웨어의 규모는 시간이 지남에 따라 복잡성과 유지보수 비용이 증가한다. 이로 인해 최근 유지보수의 중요성이 더욱 대두되고 있다. 소프트웨어가 진화 할수록 유지보수를 어렵게 하는 징후인 코드의 나쁜 냄새(Bad Smell)가 점점 심해지기 때문에 나쁜 냄새가 나는 코드를 제거하여 유지보수를 용이하게 개선해야 한다. 최근에는 이러한 나쁜 냄새를 위해 소프트웨어 리팩토링 기법에 대한 연구가 많이 연구되고 있다. 본 논문에서는 나쁜 냄새의 한 종류인 긴 매개변수 리스트(Long Parameter List)를 식별하고 해결하여 소프트웨어의 유지보수성을 향상시키는 방안을 제안한다. 제안되는 방안은 매개변수간의 의미적인 유사도를 측정하여 이를 군집화 하여 새로운 객체가 될 수 있는 매개변수들을 식별한다. 제안되는 방안은 경력 있는 객체지향 소프트웨어 개발자들이 군집화한 매개변수리스트와의 비교를 통해 평가되고, 그 결과가 통계적으로 검증된다.

키워드 리팩토링, 긴 매개변수 리스트, 유지보수, 품질

Abstract Complexity and maintenance cost of software increase as much as software has been evolved, therefore importance of software maintenance recently arise. There are many signs that are difficulties to maintain software, called bad smell, in a large-scale software. The bad smell should be removed to improve maintainability. Recently, many software refactoring methods have researched to terminate the bad smell. In this paper, we propose how to identify long parameter list, which causes bad smell, and how to solve the problem for increasing software maintainability. In our approach, we classify the parameters for creating new objects by measuring semantic similarity among them. This is evaluated by experienced software developers, and the result is statistically verified.

Key words Refactoring, Long Parameter List, Software Maintenance, Software Quality

† 이 논문은 2013년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임 (No. 2012M3C4A7033348).

‡ 비 회 원 : 서강대학교 컴퓨터공학과
donghwa@sogang.ac.kr
juna@sogang.ac.kr
§ 정 회 원 : 서강대학교 기술경영전문대학원 조교수
psjdream@sogang.ac.kr
¶ 종신회원 : 서강대학교 컴퓨터공학과 교수
sypark@sogang.ac.kr

1. 서 론

오늘날의 소프트웨어는 과거에 비해서 그 규모와 복잡성이 증가하고 있다[1]. 이에 따라 유지보수 비용 또한 증가하는데, 이때의 비용이 전체 비용의 67%를 차지한다[2, 3]. 따라서 소프트웨어의 유지보수성을 향상시켜 변경하기 용이한 코드로 개선시킬 필요가 있다. 소프트

웨어의 유지보수성을 향상시키기 위한 방법 중 하나로 리팩토링을 사용할 수 있다[4-7]. 리팩토링이란 기능은 그대로 유지한 채, 소스코드를 읽기 쉽고 수정하기 편하도록 내부를 수정하는 작업으로 정의될 수 있다[2]. M. Fowler는 소스코드에서 리팩토링의 대상이 될 수 있는 부분을 Bad Smell이라 정의했는데, 이는 소스코드에서 잠재적으로 문제를 발생시킬 수 있는 부분을 말한다[2].

코드의 나쁜 냄새 중 하나로 메소드에 선언된 매개변수가 과도하게 많을 때 이것을 긴 매개변수 리스트(Long Parameter List)라 정의하였다[2]. 매개변수 리스트가 길어질수록 코드의 가독성을 떨어뜨리기 때문에 코드를 이해하고 유지보수하기 어려워진다. 이러한 이유로 긴 매개변수 리스트를 식별하고, 이를 간소화하는 리팩토링이 필요하다.

M. Fowler는 긴 매개변수 리스트를 리팩토링하기 위해 매개변수를 객체로 전하는 방안을 소개하였다[2]. 이는 의미적으로 유사한 매개변수를 하나의 객체로 묶어서 관리하는 방법으로서, 메소드의 가독성을 증가시켜 유지보수를 용이하게 한다. 그러나 규모가 큰 소프트웨어의 경우 모든 메소드의 매개변수리스트를 조사하여 그들 간의 의미적인 유사성을 판단하고 객체로 묶는 작업은 시간과 노력이 많이 드는 작업일 뿐만 아니라, 사람에 따라 의미적인 유사성을 달리 판단할 수도 있다[2].

본 논문에서는 긴 매개변수 리스트문제를 해결하기 위해 매개변수 사이의 의미적 유사도를 측정하여 긴 매개변수 리스트를 식별하고, 군집화하는 방안을 제안한다. 제안하는 방법은 두 단계로 이루어진다. 첫 번째 단계에서는 대상으로 하는 전체 메소드의 매개변수의 식별자를 추출한 후 모든 매개변수 쌍에 대해 의미적 유사도를 측정한다. 두 번째 단계에서는 첫 번째 단계에서 측정한 의미적 유사도를 기반으로 긴 매개변수 리스트를 식별하고, 이에 대해 군집화 한다. 본 논문에서는 소프트웨어의 모듈화를 목적으로 하는 다양한 연구에서 그 효용성이 검증된 계층적 응집 군집화 알고리즘을 사용한다[8]. 계층적 응집 군집화 알고리즘은 상향식 군집화 알고리즘으로써 가장 유사한 군집간의 병합을 통해 군집화를 수행한다[9].

본 논문의 구성은 다음과 같다. 2장에서는 긴 매개변수리스트를 리팩토링하기 위해 M. Fowler가 소개한 방안에 대하여 설명하고, 이 방안들의 문제점을 기술한다. 3장에서는 긴 매개변수 리스트를 정의하고, 본 논문에서 문제를 해결하기 위해 제안한 접근방법을 기술한다. 4장에서는 본 논문에서 제안한 접근방법을 이용하여 세 개의 오픈소스 시스템을 대상으로 실험을 진행한 후 그 결과를 평가한다. 마지막으로 5장에서는 본 논문의 결론과 향후 연구가 기술된다.

2. 관련 연구

다른 나쁜 냄새에 비해 긴 매개변수 리스트는 거의 연구된 바가 없다. 따라서 본 장에서는 M. Fowler가 처음에 소개한 긴 매개변수 리스트를 해결하기 위한 리팩토링 기법을 설명하고, 이 기법들의 특징과 한계를 기술한다. M. Fowler가 소개한 3가지 리팩토링 기법은 다음과 같다[2].

1) 메소드로 전환(Replace Parameter Method)
2) 객체 전체를 전달(Preserve Whole Object)
3) 매개변수를 객체로 전환
   (Introduce Parameter Object)

첫 번째는, 어떤 메소드 B가 메소드 A의 수행결과를 매개변수로 사용할 때, 메소드 B의 구현부에서 메소드 A를 직접 호출할 수 있으면, 그 매개변수를 제거하고 메소드 B의 구현부에서 메소드 A를 직접 호출하도록 변경하는 리팩토링

기법이다. 두 번째는, 특정 객체로부터 가져온 데이터를 매개변수로 사용할 때, 기존의 매개변수를 그 객체로 대체하는 리팩토링 기법이다. 마지막은, 다수의 매개변수를 하나 혹은 여러 객체로 전환하는 리팩토링 기법이다. 첫 번째와 두 번째 리팩토링 기법은 특정 상황을 만족해야 하기 때문에 일반적으로 사용하기 어렵다. 반면에, '매개변수를 객체로 전환' 리팩토링 기법은 상황에 상관없이 사용할 수 있지만, 어떻게 다수의 매개변수를 객체로 전환할지는 여전히 개발자의 주관적인 판단에 의존한다[2]. 즉, 여러 개의 매개변수를 하나의 객체로 전환할지, 다수의 객체로 나누어 전환할지를 판단하는 기준은 개발자 마다 다를 수 있다. 따라서 본 논문에서는 긴 매개변수 리스트를 정형화 하여 정의하고, 이를 식별하고 간소화하기 위한 방안을 제시한다.

3. 접근 방법

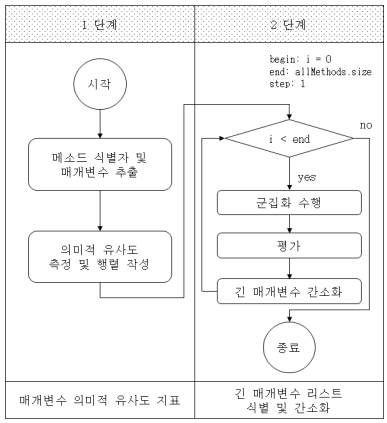
본 장에서는 긴 매개변수 리스트를 정의하고, 정의된 긴 매개변수리스트를 식별하고 이를 군집화 하여 매개변수의 수를 줄이기 위한 방안을 기술한다.

M. Fowler는 하나의 메소드에서 사용하는 매개변수가 과도하게 많을 때 이것을 긴 매개변수 리스트라 정의하였는데[2], 이는 주관적으로 해석될 수 있다. 본 논문에서는 긴 매개변수 리스트를 매개변수 리스트 중 한 쌍 이상의 매개변수가 의미적으로 관련되어 있어 하나의 클래스로 그룹화 되어 관리되어야 할 필요가 있는 매개변수 리스트라고 정의한다. 더 정형화한 정의를 위해 메소드 m의 매개변수의 집합을 $P_m = \{p_0, p_1, \ldots, p_i\}$라 하자. 그러면 의미적으로 유사한 매개변수 쌍의 집합 SSP_m은 아래와 같이 정의될 수 있다.

$$SSP_m = \{(p_i, p_j) | p_i \neq p_j \wedge SS(p_i, p_j) > \tau\}$$

의미적으로 유사한 매개변수의 집합이 $SSP_m \neq \Phi$인 조건을 만족할 때, 메소드 m의 매개변수집합 P_m은 긴 매개변수 리스트로 규정될 수 있다. 이 때 의미적인 유사도 $SS(p_i, p_j)$는 다음 장에서 자세히 기술 되며, 임계치 τ는 실험을 통해 경험적으로 결정된다.

[그림 1]는 긴 매개변수 리스트를 식별하기 위한 전체적인 프로세스를 보여준다.



[그림 1] 긴 매개변수 리스트 문제를 해결하기 위한 프로세스

본 논문에서 제안하는 방법은 크게 2단계로 나눈다. 1단계에서는 모든 메소드에 대해서 매개변수의 식별자를 추출하고 그들 간의 의미적인 유사도를 측정한다. 의미적인 유사도는 추출된 매개변수의 식별자가 같은 메소드 내에서 동시에 변수(local variable, parameter)로 선언될 조건부 확률로 정의된다. 매개변수 쌍의 식별자와 같은 식별자를 가지는 변수가 같은 메소드 내에 많이 선언될수록 그 매개변수 쌍의 의미적인 유사도는 높다. 2단계에서는 전 단계에서 측정된 의미적 유사도를 이용하여 군집화를 수행하고, 군집화의 결과로 생성된 군집의 수와 형태에 따라 긴 매개변수 리스트를 식별한다.

3.1 매개변수 의미적 유사도 지표

매개변수의 의미적 유사도를 측정하기 위해 매개변수의 식별자가 같은 메소드 내에서 얼마나 같이 등장하는지에 대한 정도를 측정하였다. 메소드는 한 가지 목적을 위해서 작성되기 때문에 [10], 메소드 내에 식별자 쌍이 동시에 많이 등장한 다는 것은 그 식별자 쌍으로 선언되어 있는 매개변수가 같은 목적을 가지고 사용된다는 것을 보여준다. 따라서 같은 메소드 내에서 많이 등장하는 식별자를 가지는 매개변수 쌍일수록 그 매개변수 쌍의 의미적 유사도는 높다고 할 수 있다. 아래의 그림 2는 본 논문에서 제안하는 방법을 설명하기 위해 임의로 작성한 예제 소스코드이다.

```
public void foo1(int a, int, b, int c) {
    a++;
    b = 10;
    c = 20;
}
public void foo2() {
    int b, c;
    b = 50;
    c = 100;
}
public void foo3() {
    int a = 10;
}
```

[그림 2] 식별자 추출을 위한 소스코드 예제

첫 번째 메소드인 foo1 메소드의 세 개의 매개변수의 식별자는 a, b, c이다. 이 매개변수의 식별자간의 의미적인 유사도를 추출하기 위해 전체 (foo1, foo2, and foo3) 메소드 내에 선언된 변수의 식별자를 아래와 같이 추출하였다.

<표 1> 추출한 메소드 식별자

메소드	식별자
foo_1	a, b, c
foo_2	b, c
foo_3	a

foo1 메소드의 식별자는 매개변수로 받아 사용한 a, b, c이고, foo2 메소드의 식별자는 지역변수로 선언하여 사용한 b, c이다. 마지막으로, foo3의 식별자는 지역변수로 선언하여 사용한 a이다.

이를 이용하여 두 매개변수 의미적 유사도는 아래와 같이 측정될 수 있다. 의미적인 유사도의 정형화된 정의를 위해서 매개변수 a의 식별자가 포함된 메소드의 집합을 M_a라고 하고 정의하면 두 매개변수 p_i, p_j의 의미적 유사도 $SS(p_i,p_j)$는 Bayes' Rule[11]을 이용하여 아래와 같이 정의될 수 있다.

$$SS(p_i,p_j)= Max\left[\frac{|M_{pi}\cap M_{pj}|}{|M_{pi}|},\frac{|M_{pi}\cap M_{pj}|}{|M_{pj}|}\right]$$

두 매개변수 사이의 의미적 유사도는 전체 메소드에서 동시에 두 매개변수의 식별자를 이용하여 선언된 변수가 전혀 없을 때 최소값을 가지며, 반대의 경우 최대값을 가진다.

위의 소스코드를 예로 들어, foo1 메소드의 매개변수 a와 b의 의미적 유사도는 다음과 같이 측정된다. 첫 번째로, 전체 메소드 중에서 a라는 식별자가 등장하는 메소드의 수를 측정한다. 위의 표에 따라 2개의 메소드(foo1, foo3)에서 a라는 식별자가 등장함을 알 수 있다. 마찬가지로, 전체 메소드 중에서 b라는 식별자가 등장하는 메소드의 수를 측정한다. 위의 표에 따라 2개의 메소드(foo1, foo2)에서 b라는 식별자가 등장함을 알 수 있다. 마지막으로, 전체 메소드 중에서 a와 b라는 식별자가 동시에 등장하는 메소드의 수를 측정한다. 위의 표에 따라 1개의 메소드(foo1)에서만 두 개의 식별자가 동시에 등장함을 알 수 있다. 이 때 매개변수의 의미적 유사도의 정의에 따라 a와 b의 의미적 유사도는 0.5(1/2)로 측정될 수 있다.

3.2 긴 매개변수 리스트 식별

본 논문에서는 긴 매개변수 리스트 문제를 해결하기 위해 매개변수 사이의 의미적 유사도를 이용한 군집화 방법을 제안한다. 군집화 알고리즘으로써 계층적 응집 군집화 알고리즘[8, 9]을 사용하였는데, 이 알고리즘은 소프트웨어의 모듈화를 목적으로 하는 다양한 연구에서 그 효용성이 검증되었다[8]. 본 논문에서는 군집간의 유사도 대신 의미적 유사도를 사용하여 군집화를 수행하고 특정 임계치를 지정하여 적절한 시점에 군집화를 중단하도록 한다.

3.2.1 의미적 유사도를 이용한 군집화

본 논문에서 제안하는 접근방법은 계층적 응집 군집화 알고리즘을 이용하여 긴 매개변수 리스트를 군집화 한다. 계층적 응집 군집화 알고리즘은 여러 군집간의 유사도를 비교하여 상향식으로 군집화를 수행하는 알고리즘이다. 본 논문에는 군집간의 유사도 대신에 매개변수 사이의 의미적 유사도를 기반으로 계층적 응집 군집화 알고리즘을 수행한다. 또한 계층적 응집 군집화 알고리즘은 최종적으로 하나의 군집만 남기 때문에, 본 논문에서는 특정 임계치를 지정하여 적절한 시점에 군집화를 중단하도록 한다.

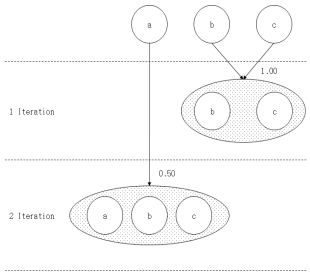
1) 계층적 응집 군집화 알고리즘

계층적 응집 군집화 알고리즘은 대상으로 하는 군집 중에서 유사도가 가장 높은 2개의 군집을 선택하여 새로운 군집을 생성하고 새롭게 생성한 군집과 기존 군집간의 유사도를 재 측정하여 군집화를 한다. 이때 새로 생성한 군집과 기존 군집 사이의 유사도를 계산하기 위하여 갱신 규칙 함수(Update rule function)를 사용하는데, 대표적으로 3가지 규칙 함수(i.e., 단일연결, 완전연결, 평균연결)가 있다[8].

본 논문에서는 평균 연결 함수를 사용하여 유사도를 계산하였다. 다음 표 2과 그림 3은 평균연결 함수를 적용한 계층적 응집 군집화 알고리즘을 사용한 예이다. 표 2는 위 소스코드에서 foo1 메소드가 사용하는 매개변수간의 의미적 유사도 행렬이고, 그림 3는 군집화 과정을 그림으로 표현한 것이다.

<표 2> foo_1 메소드의 매개변수 간에 의미적 유사도 행렬

foo_1	a	b	c
a	-	0.5	0.5
b	-	-	1
c	-	-	-



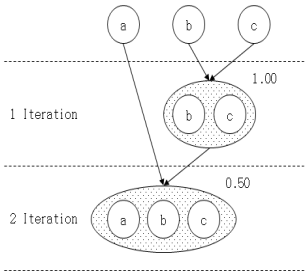
[그림 3] 계층적 응집 군집화 알고리즘의 수행결과

foo1 메소드를 대상으로 군집화하면 총 2단계로 수행된다. 첫 번째 단계는, 매개변수 중 가장 의미적 유사도가 높은 두 매개변수를 선택하여 새로운 군집을 만든다. 그러므로 1의 의미적 유사도를 가지는 b와 c를 선택하여 군집을 만든다. 다음 단계로, 새로 생성된 군집과 기존 군집간의 의미적 유사도를 평균연결 함수로 재 측정한다. a와 b, a와 c의 의미적 유사도를 계산하면, 각각 0.5이며 평균은 0.5이다. 이 값을 토대로 새로 생성한 군집과 b를 하나로 합친다. 결과적으로 3개의 매개변수로 구성된 군집 하나가 생성되고 이 군집의 의미적 유사도는 0.5이다.

위의 예와 같이 계층적 응집 군집화 알고리즘을 수행하면, 최종적으로 하나의 군집만 남는다. 즉, 반복수행의 중단시점을 결정하지 않으면 항상 하나의 군집만 남는 문제가 발생한다. 그러므로 본 논문에서는 적절한 시점에 군집화 수행을 중단하기 위하여 임계치를 이용하였고, 이는 실험을 통해서 선택하였다.

2) 임계치를 이용한 군집화 중단 지점 결정

계층적 응집 군집화 알고리즘을 그대로 적용할 시 최종적으로 하나의 군집만 남는다는 문제가 발생한다. 이를 해결하기 위해서 본 논문에서는 실험을 통해서 선택해 최선의 임계치를 지정하여 적절한 시점에 군집화 수행을 중단하도록 한다. 임계치를 지정하면, 매 반복 단계마다 군집간의 의미적 유사도와 임계치를 비교하여, 임계치 보다 큰 의미적 유사도를 가질 경우, 새로운 군집을 생성한다. 만약 임계치보다 높은 의미적 유사도를 가지는 군집이 존재하지 않으면 군집화를 중단한다. 임계치는 0.0부터 1.0까지 지정 가능하다. 0.0으로 지정하면 기존의 알고리즘 수행과 동일하므로 최종적으로 하나의 군집만 남는다. 반대로 1.0을 지정하면 임계치보다 높은 군집이 존재하지 않으므로 군집화를 중단하여, 결과적으로 매개변수의 수와 동일한 군집이 남는다. 아래의 [그림 4]는 [그림 3]의 예를 수정하여, 임계치를 지정해 적절한 시점에 군집화를 중단하도록 한 결과이다.



[그림 4] 임계치를 적용한 계층적 응집 군집화 알고리즘 수행결과

임계치 0.7을 지정하여 foo1 메소드를 대상으로 군집화하면 총 2단계로 수행한다. 첫 번째 단계는, 임계치 0.7 보다 큰 의미적 유사도를 가지는 b와 c를 선택하여 군집을 생성한다. 다음 단계로, 새로 생성한 군집과 기존 군집간의 의미적 유사도를 평균연결 함수로 재 측정한다. a와 b, a와 c의 의미적 유사도는 각각 0.5과 0.5로 측정된다. 두 경우 모두 임계치 0.7보다 작으므로 군집화를 중단한다. 결과적으로 b와 c를 포함하는 하나의 군집과 a를 포함하는 군집 하나, 총 두 개의 군집이 생성된다. 앞서 정의한 긴 매개변수 리스트의 정의에 따라 foo1 메소드는 긴 매개변수 리스트로 식별된다.

3.3 긴 매개변수 리스트 간소화

이렇게 군집화한 매개변수를 간소화 위해 '매개변수를 객체로 전환(introducing parameter object)' 기법을 사용한다. 이 기법은 두 가지 단계를 거쳐 수행한다. 첫 번째 단계는 하나로 합칠 매개변수를 멤버변수로 가지는 새로운 클래스를 정의한다. 두 번째로, 메소드의 매개변수를 새로 생성한 클래스로 대체하고, 구현부를 그에 맞게 수정한다. 리팩토링의 결과는 아래의 [그림 5, 6]과 같다.

```
public void foo1(int a, ParamObj paramObj) {
    a++;
    paramObj.setB(10);
    paramObj.setC(20);
}
```

[그림 5] '매개변수를 객체로 전환' 기법을 적용한 매개변수 대체

```
public class ParamObj {

    private int b;
    private int c;

    public ParamObj (int b, int c) {
        this.b = b;
        this.c = c;
    }
    public int getB() {
        return b;
    }
    public void setB() {
        this.b = b;
    }
    public int getC() {
        return b;
    }
    public void setC() {
        this.b = b;
    }
}
```

[그림 6] 매개변수 b와 c를 포함하는 새로운 클래스

제안하는 군집화 방법의 결과로써, 매개변수 b와 c는 의미적으로 유사하기 때문에, 두 매개변수를 하나의 객체로 전환할 수 있다. 긴 매개변수 리스트를 간소화하기 위해 첫 번째로, 매개변수 b, c와 동일한 멤버변수를 가지는 ParamObj 클래스를 [그림 6]처럼 새로 정의한다. 두 번째로, foo1 메소드가 사용하는 b와 a 매개변수를 ParamObj 클래스 타입의 paramObj 객체로 전환한다. 마지막으로, [그림 5]와 같이 paramObj 객체의 Getter, Setter를 이용하여 b와 c 데이터에 접근할 수 있도록 foo1 메소드를 수정한다. 위와 같이 본 논문에서 제안하는 군집화 방법을 이용하면, 긴 매개변수리스트를 식별하고, 의미적으로 유사한 매개변수를 하나의 객체로 전환하여 긴 매개변수 리스트를 간소화 시킬 수 있다.

4. 평 가

본 장에서는 제시되는 방안의 평가 계획, 평가 시행방법, 평가 결과가 기술된다.

4.1 평가 계획

제시되는 방안은 3가지 오픈소스 시스템(JHotDraw, JEdit, ArgoUML)을 대상으로 평가되었다. 평가 대상 시스템으로 사용하는 오픈소스의 정보는 다음과 같다.

<표 3> 대상 시스템 정보

	JHotDraw	JEdit	ArgoUML
개발 언어	Java	Java	Java
클래스 수(개)	772	1,612	1,084
라인 수(LOC)	82,099	122,054	117,370
메소드 수(개)	7,297	10,239	7,412

JHotDraw[12]는 자바 기반의 드로잉 프로그램이고 JEdit은 텍스트 편집 프로그램[13], Argo UML은 UML 모델링 도구[14]로서 각기 다른 도메인과 규모를 가지는 시스템이다. 이 시스템을 대상으로, 제시되는 방안을 통해 식별된 긴 매개변수 리스트 중 아래의 조건을 만족하는 긴 매개변수 리스트를 추출하였다.

1) 이미 선정한 메소드의 매개변수의 식별자와 동일한 식별자의 매개변수를 가지는 메소드는 선정에서 제외한다. 그 이유는, 동일한 이름의 매개변수를 사용하는 메소드는 이전과 비슷한 매개변수의 군집 형태를 가질 수 있기 때문이다.
2) Getter/Setter, 생성자는 선정에서 제외한다. 이는 클래스의 변수(Attribute)만을 다루는 메소드이기 때문이다.

추출된 긴 매개변수 리스트를 가지는 메소드들은 아래의 군집화 품질지표를 이용하여 군집화 후의 품질이 측정되었다.

$$CQ(m)=\underset{p_i\in P_m, p_k\in P_m}{\arg max} SS(p_i,p_k)$$

위의 수식을 이용하여, 모든 메소드 m에 대해 m이 가지는 매개변수의 집합 P_m내의 매개변수 쌍 중에 높은 의미적 유사도를 가지는 메소드 순으로 우선순위화 하고 프로젝트 별로 상위 10개씩 추출하였다. 추출된 30개의 메소드는 아래와 같이 선정된 전문가 그룹이 군집화 한다. 전문가 그룹은 자바 개발 실무경험이 있는 세 명(석사 학위를 가지는 10년 이상의 개발자 X 2명, 학사 학위를 가지는 3년 이상의 개발자 X 1명)으로 구성되어 있다. 전문가 그룹은 주어진 메소드의 내용을 파악하여 의미적으로 유사한 매개변수를 군집화 하였고, 전문가 그룹이 군집화한 결과와 제시되는 방안을 통해 자동으로 군집화된 결과가 일치하는 정도를 아래의 $FMeasure$를 기반으로 평가하였다.

$FMeasure$의 정의를 위해 M_{LPL}을 긴 매개변수 리스트를 가지는 메소드의 집합이라 칭하고, M_{LPL}의 원소인 m은 여러 개의 유사한 매개변수의 집합 C_i로 군집화 될 때, 집합 C_i를 원소로 가지는 C_i의 멱집합을 $Clst_m=\{c_0,c_1,\ldots,c_i\}$ 이라고 하자. 그러면 메소드 m에 대한 $FMeasure(m)$은 아래와 같이 정의된다.

- $P(c_i) = \frac{|c_i \cap c'_i|}{|c_i|}$
- $R(c_i) = \frac{|c_i \cap c'_i|}{|c'_i|}$
- $FMeasure(c_i) = 2 * \frac{P(c_i) * R(c_i)}{P(c_i) + R(c_i)}$
- $FMeasure(m) = \frac{\sum_{c_i \in Clst_m} FMeasure(c_i)}{|Clst_m|}$

위의 정의에서 c'_i는 전문가가 군집화한 매개변수를 말한다.

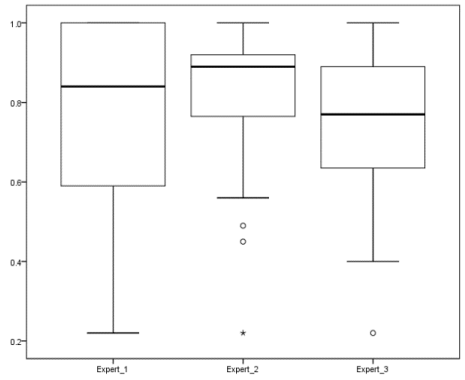
4.2 평가 시행

본격적인 평가에 앞서, 평가에 대한 기본적인 내용을 설명하고, 추출된 30개의 긴 매개변수 리스트를 배포하였다. 전문가 그룹은 긴 매개변수 리스트를 가지는 메소드 내에 있는 여러 정보를 토대로 매개변수를 군집화하고, 자신이 판단한 매개변수의 군집 형태를 그림으로 표시하여 제출한 한다.

한 전문가가 다른 전문가와 평가에 대하여 정보를 공유할 경우, 해당 전문가 간에 유사한 형태의 결과가 발생할 수 있기 때문에, 이를 방지하기 위해 개별적으로 평가를 진행한다. 또한 제공한 평가항목의 내용을 충분히 숙지하고 평가에 응할 수 있도록 하루의 제출기한을 두어 긴 매개변수리스트를 검토하도록 한다.

4.3 평가 결과 분석

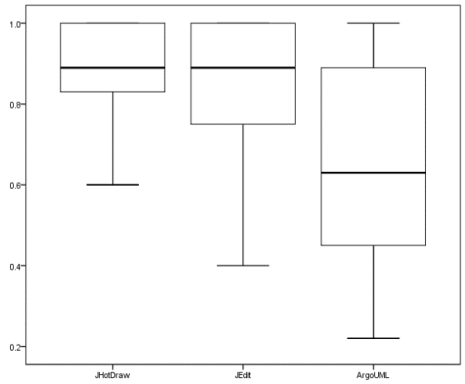
본 실험에서는 긴 매개변수 리스트의 군집화를 위해 임계치를 적용하였다 (τ = 0.5). 자동으로 군집화된 긴 매개변수의 리스트는 아래의 <표 4>와 같다. 표 4에 나타난 매개변수 군집화 결과와 전문가 그룹이 수작업으로 군집화한 결과를 비교하였다. 먼저 3명의 전문가 모두가 공통적으로 30개의 긴 매개변수 리스트 중 6개는 군집화가 필요 없다 판단하였다. 나머지 24개의 매개변수 리스트를 자동으로 군집화한 결과와 각 전문가가 수작업으로 군집화 한 결과와의 일치율을 나타내는 F-Measure의 분포는 아래와 같다.



[그림 7] 전문가 별 평가 결과

각각 세 명의 전문가가 군집화한 결과는 자동으로 군집화된 결과와 비교하여 평균적으로 0.77, 0.83, 0.74의 F-Measure로 측정되었다. 한편 전문가별로 추출한 샘플의 F-Measure가 차이가 있기 때문에 F-Measure의 유의수준 0.05 에서 통계적인 신뢰구간을 추정하였다. 추정된 F-Measure의 신뢰구간은 0.72~0.82로 추정되었는데, 이는 추출된 샘플에서 측정된 F-Measure가 평균 0.78을 보였지만 결과의 평균과 편차를 고려하였을 때, 그 평균이 95%의 확률로 0.72~0.82 구간 내에 존재한다는 것을 나타낸다.

아래의 그림은 프로젝트 별 F-Measure의 분포를 보여준다.



[그림 8] 프로젝트 별 평가 결과

#	프로젝트	메소드	군집결과
1	JH	createRect	[doc], [attributes], [x, y, width, height], [rx, ry]
2	JH	cap	[p1, p2], [radius]
3	JH	encodeBytes	[source], [off, len], [options]
4	JH	createCircle	[doc], [attributes], [cx, cy, r]
5	JH	reparameterize	[d, bezCurve, first, last], [u]
6	JH	draw	[g], [f], [p1, p2]
7	JH	createColorWheelChooser	[sys], [angularIndex, radialIndex, verticalIndex, flipX, flipY], [type]
8	JH	groupFigures	[view, group], [figures]
9	AG	connect	[fromPort, toPort], [edgeType]
10	AG	buildConnection	[graphModel], [edgeType], [sourceFig, destFig]
11	AG	getPointOnPerimeter	[rect], [direction], [xOff, yOff]
12	AG	fireTreeNodesChanged	[source], [path, childIndices, children]
13	AG	startElement	[uri, localname, qname, attributes]
14	AG	setBounds	[xInc, yInc, w], [concurrency]
15	JE	hbAssignCodes	[code], [minLen, maxLen, alphaSize], [length]
16	JE	paintValidLine	[gfx, y], [screenLine, physicalLine], [start, end]
17	JE	parseStyle	[str, defaultFgColor, family, size], [color]
18	JE	getWriteEncodingErrorMessage	[encodingName, encoding], [line], [lineIndex]
19	JE	fireContentRemoved	[startLine, numLines], [length], [offset]
20	JE	extendSelection	[offset], [end, extraStartVirt, extraEndVirt]
21	JE	prepareBackupFile	[path], [backups, backupPrefix, backupSuffix, backupTimeDistance, backupDirectory]
22	JE	nextMatch	[text, firstTime, reverse], [start, end]
23	JE	doFontSubstitution	[subst, start, end], [mainFont, text]
24	JE	paintScreenLineRange	[textArea], [firstLine, lastLine], [gfx, y, lineHeight]

<표 4> 긴 매개변수 리스트와 군집결과

세 프로젝트의 평균 F-Measure는 각각 0.83, 0.83, 0.69로서 JHotDraw와 JEdit이 더 높은 결과를 보였다. 프로젝트 별 F-Measure의 편차가 통계적으로 무의미하고, 프로젝트간의 차이가 없다는 것을 증명하기 위해 유의 수준을 0.05로 하여 대응 표본 T 검정을 시행 하였다. 검정 결과는 아래의 <표 5>와 같다.

표 5 프로젝트 별 대응 표본 T검정 결과

	대응차					유의 확률
	평균	표준 편차	표준 오차	차이의 95% 신뢰구간		
				하한	상한	
JH-JE	0.04	0.16	0.04	-0.04	0.11	0.295
AG-JE	-0.16	0.40	0.09	-0.34	0.02	0.077
JH-AG	0.14	0.34	0.07	0.01	0.28	0.038

위의 검정결과에서 보는 바와 같이 JHotDraw와 JEdit 그리고 ArgoUML과 JEdit의 평균의 차이는 각각 0.04, 0.16의 차이가 나지만 유의 확률이 미리 설정된 0.05보다 크기 때문에 통계적으로 차이를 인정할 수 없으며, 그러므로 두 쌍의 프로젝트 간의 평균의 차이는 의미가 없다고 할 수 있다. 그러나 JHotDraw와 ArgoUML의 경우 유의 확률이 0.05미만으로 차이가 있다고 할 수 있다. 이러한 차이는 두 프로젝트의 품질과 관계가 있다고 할 수 있다. 왜냐하면 자동으로 군집화하기 위해 사용한 매개변수의 식별자간의 의미적 유사도는 메소드 내에 선언된 변수가 많이 등장할수록 높은 값을 가지도록 설계되어 있지만, 그 결과가 전문가들이 판단하기에는 의미적 유사도가 적다고 판단하였기 때문이다.

결과적으로 이는 ArgoUML의 메소드들이 JHotDraw의 메소드들 보다 의미적인 응집력이 떨어진다고 할 수 있다. 표 3에서 보는 바와 같이 ArgoUML의 경우 제시되는 방안을 통해 식별된 10개의 긴 매개변수 리스트 중 4개(40%) 가 전문가들의 판단으로는 군집화할 수 없는 매개변수로 판별되어 잘못 식별된 것으로 평가되었는데, 이 또한 ArgoUML의 품질이 상대적으로 떨어지기 때문일 것으로 판단된다. 우리는 이러한 차이를 검증해 보기 위해 두 프로젝트 내에 사용된 텍스트를 이용한 의미적 응집력 지표인 C3[15] 지표를 이용하여 두 프로젝트의 품질을 비교하였다. 측정결과 JHotDraw의 C3는 약 0.66으로 측정된 반면 ArgoUML은 0.53으로 측정되었는데, 이러한 사실은 프로젝트의 품질이 낮을수록 본 제안방법의 정확도도 떨어진다는 사실을 뒷받침해준다.

5. 결 론

본 논문은 소프트웨어의 유지보수성을 떨어뜨리는 요소 중 하나인 나쁜 냄새의 한 종류인 긴 매개변수 리스트 문제를 해결하는 방안으로 매개변수간의 의미적 유사도를 이용한 군집화 방법을 제안했다. 본 논문에서 제안하는 방법은 세 개의 오픈소스를 대상으로 평가되었다. 평가는 제시되는 방안을 통해 식별한 긴 매개변수 리스트 30개를 대상으로 세 명의 전문가 그룹이 수작업으로 군집화하고, 이렇게 군집화한 결과와 제시되는 방안을 통해 자동으로 군집화한 결과를 비교하였다. 비교결과 평균 0.78정도의 정확도를 보였고, 통계적으로 F-Measure의 평균의 신뢰구간은 0.72~0.82내에 존재하는 것으로 추정되었다. 또한 프로젝트간의 F-Measure를 비교한 결과 ArgoUML이 가장 정확도가 떨어지는 것으로 나타났다. 텍스트 기반의 클래스 응집도 지표를 기반으로 ArgoUML의 품질을 측정해 본 결과 ArgoUML의 품질이 다른 시스템에 비해서 떨어지기 때문인 것으로 나타났다. 따라서 본 논문에서 제안되는 방안은 메소드의 의미적인 응집도가 너무 낮으면 정확도가 떨어지는 한계를 가지고 있다고 할 수 있다. 향후 연구로서 메소드의 의미적인 응집도를 고려하여 정확도를 향상시키기 위한 연구를 진행해 보고자 한다.

참 고 문 헌

[1] Y. Ren, T. Xing, X. Chen, X. Chai, “Research on Software Maintenance Cost of Influence Factor Analysis and Estimation Method”, 2011 3rd International Workshop on Intelligent Systems and Applications, pp.1-4, May 2011

[2] M. Fowler, “Refactoring: Improving the Design of Existing Code”, Addison Wesley, 1999

[3] C. Zhao, J. Kong, K. Zhang, “Program Behavior Discovery and Verification: a Graph Grammar Approach”, IEEE Transactions on Software Engineering, Vol.32, Issue.3, pp.431-448, May-June 2010

[4] M. Arora, Dr. S. S. Sarangdevot, Vikram Singh Rathore, J. Deegwal , S. Arora, “Refactoring, Way for Software Maintenance”, IJCSI International Journal of Computer Science Issues, Vol.8, Issue.2, March 2011

[5] Kataoka. Y, Imai. T, Andou. H, Fukaya T, “A Quantitative Evaluation of Maintainability Enhance ment by Refactoring”, Proceedings of the International Conference on Software Maintenance, pp.576-585, 2002

[6] Eva van Emden, Leon Moonen, “Java Quality Assurance by Detecting Code Smells”, Proceedings of the Ninth Working Conference on Reverse Engineering, pp.97-106, 2002

[7] Meananeatra. P, “Identifying Refactoring Sequences for Improving Software Maintainability”, Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, pp.406-409, Sept 2012

[8] O. Maqbool, H.A. Babri, “Hierarchical Clustering for Software Architecture Recovery”, IEEE Transactions on Software Engineering, Vol.33, Issue.11, pp.759-780, Nov 2007
[9] O. Maqbool, H.A. Babri, “The Weighted Combined Algorithm: A Linkage Algorithm for Software Clustering”, Proceedings of the Eighth European Conference on Software Maintenance and Reengineering, pp.15-24, March 2004
[10] Robert C.Martin, “Clean Code: A Handbook of Agile Software Craftsmanship”, Prentice Hall, 2008
[11] T. Bayes, R. Price, “An Essay towards solving a Problem in the Doctrine of Chances. By the late Rev. Mr. Bayes, communicated by Mr. Price, in a letter to John Canton, M. A. and F. R. S.”, 1763
[12] JHotDraw, http://www.jhotdraw.org/
[13] JEdit, http://www.jedit.org/index.php
[14] ArgoUML, http://argouml.tigris.org/
[15] Marcus. A, Poshyvanyk. D, “The conceptual cohesion of classes”, Proceedings of the 21st IEEE International Conference on Software Maintenance, pp.133-142, Sept 2005

저자 소개

함 동 화

2012년 명지대학교 컴퓨터공학과(학사)
2013년~ 서강대학교 컴퓨터공학과 석사과정
관심분야는 소프트웨어 유지보수, 소프트웨어 품질

이 준 하

2007년 단국대학교 컴퓨터과학과(학사)
2010년 서강대학교 컴퓨터공학과(석사)
2010년~ 서강대학교 컴퓨터공학과 박사과정
관심분야는 소프트웨어 유지보수, 소프트웨어 품질

박 수 진

1995년 경북대학교 컴퓨터공학과(학사)
2000년 서강대학교 정보통신대학원(석사)
2008년 서강대학교 컴퓨터공학과(박사)
1995년~1999년 (주)LG-CNS 근무
2000년~2002년 한국래쇼날 소프트웨어 선임 컨설턴트
2010년~현재 서강대학교 기술경영전문대학원 조교수
관심분야는 소프트웨어 재사용, 요구공학, 소프트웨어 아키텍처, 적응형 소프트웨어

박 수 용

1986년 서강대학교 컴퓨터과학(학사)
1988년 Florida State University, 컴퓨터정보과학(석사)
1995년 George Mason University, 정보기술(박사)
1998년~서강대학교 컴퓨터공학과 교수
관심분야는 소프트웨어 요구사항, 자가적응형 소프트웨어, 소프트웨어 품질