소프트웨어 재사용을 위한 실증적 객체지향 Refactoring 프로세스 설계

박진호° 이종호* 류성열

숭실대학교 컴퓨터학과, 대신정보통신㈜*
jhpark@selab.soongsil.ac.kr, jhlee@dsic.co.kr, syrhew@computing.soongsil.ac.kr

A Study of Positive Object-Oriented Refactoring Technique for the Software Reuse

Jin-Ho Park° Jong-Ho Lee* Sung-Yul Rhew

Dept. of Computing, SoongSil University, Daishin Information & Communications Co., LTD.*

요 약

현재 기업들은 거의 모든 부분이 전산화 작업으로 이루어지고 있으며, 기업의 중요한 업무는 모두 방대한 시스템이 사람을 대신해서 처리해 주고 있는 실정이다. 하지만 시스템 유지보수의 체계적인 프로세스에 대해서는 현재 연구된 바가 없어 쉽게 적용할 수 없는 문제가 존재한다. 재공학은 재사용 하려는 소프트웨어를 분석, 재정의, 재문서화 등의 소프트웨어 역공학 방법을 통하여 좀더 효율적으로 소프트웨어를 분석할 수 있고, 순공학을 통해 문제점들의 교환과 새로운 구조와 시스템의 수정을 통해 더욱 경제적인 시스템을 만들 수 있다. 객체지향 어플리케이션의 Refactoring에서는 대표적으로 클래스간의 공통성을 추출하고, 이를 일반화 시켜 추상클래스를 생성 및 삭제, 이동하는 과정을 통해 소프트웨어의 구조를 변경시켜 시스템의 성능향상을 도모하게 된다. 본 논문에서는 시스템의 재사용을 위한 실증적인 객체지향 Refactoring 기법을 제시한다.

1. 서 론

소프트웨어의 구조를 변경시키는 작업은 눈에 잘 드러나지 않으면서도 심각한 문제를 초래할 수 있으며, 그 대상을 현재 운영중인 소프트웨어를 대상으로 하기 때문에 소프트웨어 설계 패턴을 기반으로 작업 대상 코드를 명확히 식별하고, 체계적으로 작은 단위씩 실행하여야 한다[1].

시스템 사용과 중요성의 비중 증가, 관리와 유지보수, 재사용을 하는데 있어서 정확한 분석 작업을 필요로 하게 되었다. [2]

소프트웨어 Refactoring 기법을 사용하면 소프트웨어 설계의 능률 향상, 이해하기 쉬운 소프트웨어 생성, 버그 검색의 용이, 소프트웨어의 성능향상 등의 이점을 얻을 수 있으며, 새로운 기능의 추가가 용이하다. 소스코드를 살펴볼 때에도 많은 도움을 준다.

본 논문은 소프트웨어를 Refactoring 할 때에 실증적으로 사용할 수 있는 프로세스를 제안하고, 그 단계들을 정의하여 사용자들이 시스템을 쉽고 편리하게 유지보수, 관리, 재사용 할 수 있는 실증적 객체지향 Refactoring 기법, 즉 POOR 기법을 제시한다.

2. 관련 연구

2.1 컴포넌트 기반 소프트웨어 개발 (CBSD)

Component-Based Software Development은 새로운 시스템을 개발하는 경우 기존의 유사한 시스템에서 개발한 컴포넌트들을 이용하여 어플리케이션을 개발하는 것을 말한다. 이러한 컴포넌트 기반의 소프트웨어를 개발하기 위해서는 프로세스 모델들 가운데 컴포넌트 조합 모델(Component Assembly Model)과 같은 방법을 이용할 수 있다[3]. 이 모델은 기존의 프로세스 모델들 가운데 하나인 나선형 모델과 같이 계획, 위험 분석, 개발 및 보급, 고객 평가의 과정을 거치지만, 새로운 소프트웨어 개발 과정에서 컴포넌트를 기본적으로 재사용하여 개발한다는 점이 다르다. 구분되어지는 과정을 살펴보면 후보 컴포넌트들을 식별 / 컴포넌트 라이브러리 조사, 유용한 컴포넌트 추출 / 새로운 컴포넌트 개발, 새로운 컴포넌트 추가 / 새로운 통합 시스템 구축의 순서이다.

컴포넌트 기반의 소프트웨어 개발의 이점은 기존의 소프트웨어와 시스템을 개발하는데 소요되는 시간과 경제적 지출을 최대한 줄일 수 있다는 것이다.

2.2 소프트웨어 재공학

소프트웨어 재공학은 자동화 도구를 이용하여 기존의 시스템을 평가하거나 수정하는 활동이다. 재공학의 목적은 유지보수성, 생산성, 품질의 향상 등을 포함한다. 일반적으로 재공학은 데이터 이름의 변경이나 정의, 프로세스 로직의 재구성과 같은 형식의 변경을 포함하지만, 기본적으로는 기능성은 변하지 않은 상태로 형식이나 구조만 변경된다. 그러나 사용자의 요구가 전문화 되고, 초 스피드화 되는 현실에서는 기본적인 구조에 새로운 기능성을 추가하는 작업이 요구되는데, 이때에는 재설계 과정까지 포함한 재공학이 이루어진다. 즉, 재공학은 소프트웨어 유지보수 과정을 향상시키고 새로운 기술과 도구를 유지보수에 적용함으로써 기존의 시스템의 기능을 향상시키는 활동을 포함한다. 일반적인 재공학은 역공학 + Change + 순공학의 의미를 가진다[4, 5, 6].

재공학의 대표적인 기법에는 재구성(Restructuring), 데이터 재공학, Refactoring 등이 있다[7].

2.3 OO-Refactoring

Refactoring이란 내부적인 구조의 개선이 되지 않거나 외부적인 코드의 행위가 너무 오래 되었거나 할 때에 소프트웨어 시스템의 프로세스들을 교환하여 성능을 향상시키거나, 기능적인 행동을 보장하면서 근본적인 원인들만 찾아내서 변화 시킴으로써 더욱더 향상된 시스템을 만들어 주는 것을 말한다[8].

- Serge Demeyer와 Stephane Ducassse의 Refactoring 방법을 보면 클래스 다이어그램을 중심으로 한 5단계 기법으로 다음과 같다[7].

1 단계: Create Subclass
2 단계: Move Attribute
3 단계: Move Method
4 단계: Split Method + Move Method
5 단계: Clean-up

- Martin Fowler가 주장하는 Refactoring 방법도 클래스 다이어그램을 중심으로 한 4단계이고, 작업의 수행은 다음과 같다[8].

1 단계: Extract Method
2 단계: Move Method
3 단계: Extract and Move Method 적용
(1 단계와 2 단계에서 미흡한 단계의 반복 적용)
4 단계: Replace Temp with Query

3. POOR 기법

POOR기법이란 Positive Object-Oriented Refactoring의 약자로 실증적 객체지향 Refactoring이란 뜻이다. 소프트웨어의 재사용 기법 중에서 핵심적 기술이 되는 Refactoring기법을 객체지향으로 개발되어진 기존 시스템에 실증적으로 적용을 시켜 시스템의 유지보수와 성능의 향상 가져올 수 있다. POOR는 현재 작동되는 시스템들의 노후와 유지보수와 개발에 드는 비용이 많기 때문에 이러한 경제적인 문제를 줄여보고자 하는 소프트웨어의 개발 의미와 부합되는 단어이기도 하다.

본 절에서는 POOR을 사용하여 시스템 사용자와 관리자들이 좀더 쉽게 이용하고, 경제적인 시스템의 유지보수와 재사용을 할 수 있도록 기존의 개략적인 추상화 단계만 적용이 되던 Refactoring기법을 실증적이고 구체적인 단계로 세분화 한 POOR기법을 제시한다.

3.1 POOR 프로세스

POOR의 첫번째 단계는 분석 단계이고, 두 번째 단계는 일반화 단계이고, 세 번째 단계는 PI-Refactoring단계이고, 마지막으로 네 번째 단계는 분할/이동시킨 요소들을 자연스럽게 만들어 주는 최적화 단계이다. 그림 1는 POOR 기법의 프로세스 4단계를 나타낸다.

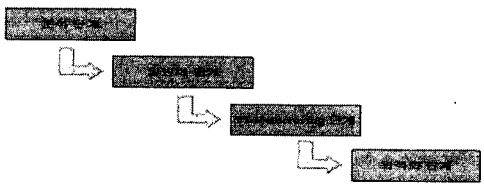

그림 1. POOR 기법 4단계

- **1단계 – 분석 단계**

: 분석단계에서는 구조와 관계, 유사도, 기능 등을 분석하여 Refactoring할 수 있는 근거 자료들을 분석해 내는 단계이다. 즉, 함수 내에 어떤 자료가 있는지의 여부와 그것들의 연관관계나 응집도, 결합도 등이 시스템에 어떤 영향을 주는 역할을 하는지를 분석하고, 불필요한 메소드와 속성들이 있는지를 찾아내는 단계이다.

- **2단계 – 일반화 단계**

: 1단계에서 찾아진 메소드와 함수들의 분석 자료를 가지고 Refactoring 작업 범위를 선정하고, 이들에 대한 Function들의 기능이나 타입(Type)들을 표준화 하여 보고서를 작성하는 단계이다.

- **3단계 – PI-Refactoring 단계**

: 2단계에서 작성된 보고서를 참조하여 각각의 요소들을 다른 클래스로 옮기거나 중복된 요소들을 삭제하는 단계이며, 메소드와 속성들이 더 많이 사용되는 함수를 찾아 기능별로 이동을 시킨다. 그 속성들이 가지고 있는 특징을 더 많이 사용하는 함수로 옮기고 이전 함수에서 호출하는 방법이다.

쿼리(Query)의 임시 저장소(Temp)와 같은 곳이 있으면 새로이 메소드와 속성들을 새로이 재배치 하여 하나의 독립된 함수로 분할하는 단계이다.

단, 슈퍼 클래스화 시키는 과정이 꼭 필요한 것은 아니며, 단순한 이동이나 중복의 삭제 만으로도 기능의 최적화를 이룰 수 있다.

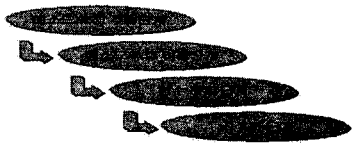

그림 2.PI-Refactoring 기법 4단계

PI-Refactoring의 첫번째 단계는 메소드(Method)/속성(Attribute) 추출 단계이고, 두 번째 단계는 메소드/속성 이동 단계이고, 세 번째 단계는 함수 분할 단계이다. 그림 2는 PI-Refactoring의 4단계를 나타낸다[9].

- **4단계 – 최적화 단계**

: 분할 되어진 함수의 메소드와 속성들을 자연스럽게 함수별로 정리하고, 수정하는 단계이다. 또한 Refactoring 되어진 함수들의 기능이 정상적으로 작동하는지의 여부와 추가적으로 향상시키려 한 기능적 요소들이 성공적으로 작업이 되었는지를 테스트 하는 단계이다.

4. POOR 프로세스의 적용

본 절에서는 앞에서 제시한 POOR기법을 이용하여 유지보수의 근거 자료 작성과 성능개선을 위한 Refactoring 과정을 A사의 금융 관련 시스템을 자료로 하여 적용과정을 보여준다.

그림 3. POOR 프로세스

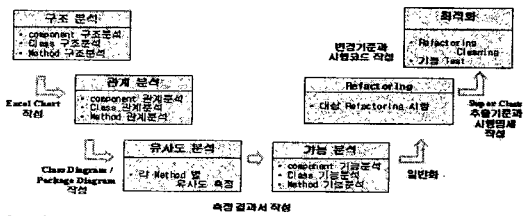

그림 3은 POOR 프로세스를 나타낸 그림이다.

- **1단계 – 분석 단계**

- 구조 분석 : 각 도메인 별 소속된 요소들을 알아내고 정리한다.
- 관계 분석 : 각 요소들 간의 관계를 나타내는 클래스 다이어그램을 만든다.
- 유사도 분석 : 자체적인 유사도 요소와 기준을 만들고, 이들의 유사도를 측정해서 %로 나타낸다.
- 기능 분석 : 실제적으로 각 함수들의 기능이 어떻게 돌아

가는지를 파악하여 형태적인 분석 뿐만이 아니라 실제로 수행되어지는 기능적인 측면까지 분석한다.

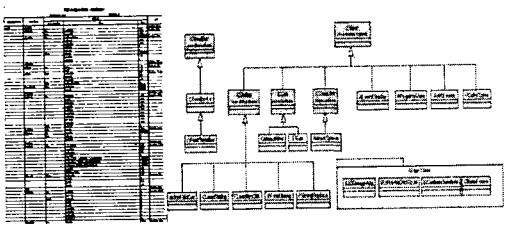

그림 4.구조분석과 관계분석 자료 산출물

그림 4 은 구조분석, 즉 시스템의 컴포넌트와 클래스, 메소드 등의 요소들이 몇 개가 있는지, 어디에 속해 있는지를 나타낸 구조분석 산출물이고, 관계분석 산출물은 구조분석에서 나온 자료들을 가지고 그들의 클래스들이 어떠한 관계가 있는지를 나타내는 클래스 다이어그램의 일부를 나타낸 것이다.

그림 5 는 기능분석을 위한 자료 소스의 정리와 그의 기능을 분석한 보고서를 나타낸 산출물이다.

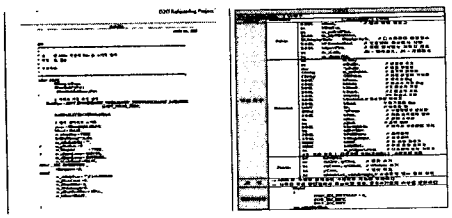

그림 5.기능분석 자료 산출물

- **2 단계 – 일반화 단계**
  : Refactoring 작업 범위로 A*와 Q*을 선정하고, Function 들의 기능, 타입(Type)들을 표준화 하여 비교된 보고서를 작성하고 단위별 일반화를 수행한다. 그림 6 은 유사도와 기능 분석 산출물을 분석해서 관계분석 산출물에 Refactoring 범위를 표시해 놓은 그림이다.

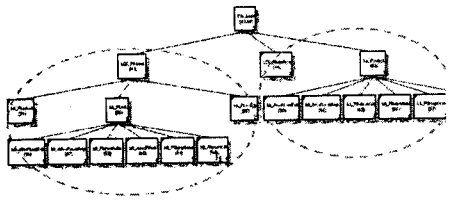

그림 6.대상 범위 선정

- **3 단계 – PI-Refactoring 단계**
  : A*와 Q*의 2 단계 보고서를 참조하여 각각의 요소들 중에서 중복되거나 유사한 요소들을 다른 클래스로 옮기거나 삭제하고, 메소드와 속성들이 더 많이 사용되는 함수를 찾아 기능별로 이동을 시킨다.
  이름이 틀리거나 중복된 요소가 없어도 기능적인 중복이 발견되면, 개발이나 관리의 책임자가 판단하여 메소드와 속성들을 새로이 재배치 하여 하나의 독립된 함수로 분할한다.
  단, 슈퍼 클래스화 시키는 과정이 꼭 필요한 것은 아니며, 단순한 이동이나 중복의 삭제 만으로도 시스템의 유지보수 비용이 감소한다. 형태적인 부분과 기능적인 부분이 모두 완료 되면 시스템의 최적화를 이룰 수 있다. 그림 7 은 Refactoring 을 이해하기 쉽고 편리하게 수행하기 위하여 만들어진 PI-Refactoring 다이어그램이다.

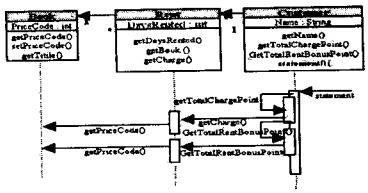

그림 7. **PI-Refactoring** 다이어그램

- **4 단계 – 최적화 단계**
  : Refactoring 되어진 A*와 Q*의 부분 중에서 통합/분할 되어진 함수의 메소드와 속성들을 자연스럽게 정리하고, 수정한다. 마지막으로 Refactoring 되어진 함수들의 기능이 정상적으로 작동하는지의 여부와 추가적으로 향상시키려 한 기능적 요소들이 성공적으로 작동이 되는지를 테스트한다.

5. 결론 및 향후 연구과제

소프트웨어의 사용이 날로 늘어가고, 경쟁적인 시스템 개발도 한계에 다다르면서 소프트웨어의 재사용의 비중이 개발을 능가하게 된다는 조사결과가 나왔다. 본 논문은 소프트웨어의 재사용 기술 중 가장 효율적인 기법으로 제시되는 객체지향 Refactoring 의 핵심적인 사항들을 정리하여 프로세스를 정의하고, 단계별 기법과 모델을 제시하였다. 새로이 제시한 POOR 기법을 이용하여 소프트웨어 유지보수와 재사용을 위해 좀더 정확하고 경제적인 방법으로 Refactoring 하는 기술을 더욱 발전시킬 수 있을 것이다.

앞으로는 더욱 다양한 분야와 전문적인 기법이 개발되어 소프트웨어 전 분야에 적용 가능한 재사용 자원들이 만들어져야 할 것이다. 프로그램의 구현도 부분적인 적용에서 벗어나, 모든 과정이 자동화 처리가 가능하여 누구나 쉽게 사용할 수 있도록 보급되는 날도 멀지 않아 가능해질 것이다.

특히, 재사용의 관심이 높아지고 있기 때문에 기능 단위의 분할이나 이동 등을 좀더 쉽게 할 수 있는 컴포넌트화 방법이나 자동화 도구의 개발 분야의 기준 제정과 연구가 더욱 진행되어야 할 것이다.

6. 참 고 문 헌

[1] Martin Fowler, "Refactoring - Improving the Design of Existing Code", Addison Wesley Longman Inc., 1999.

[2] 권오천, 신규상, "역공학 및 재공학의 기술동향", 한국정보과학회, 소프트웨어공학회지 P6~21, 1999 년 3 월 호.

[3] Adler, R. M., "Emerging Standards for Component Software," IEEE Computer, vol. 28, PP. 68~77, March 1995.

[4] Scott Tilley, "A Reverse-Engineering Environment Framework", Carnegie Mellon Software Engineering Institute, April 1998.

[5] Nicolas Anquetil and Timothy Lethbridge, "Extraction Concepts from File Names; a New File Clustering Criterion", IEEE, 1998.

[6] Ivar Jacobson, "Re-engineering of old systems to an object-oriented architecture", OOPSLA, 1991.

[7] Serge Demeyer, Stephane Ducassse, "Object-Oriented Reengineering", OOPSLA, 1999.

[8] Martin Fowler, "Refactoring: Improving the Design of Existing Code", OOPSLA, 1999.

[9] 박진호,이종호,류성열,"소프트웨어 유지보수와 재사용을 위한 재공학 Refactoring 기법 연구",한국정보과학회,춘계학술발표논문집,2000.