소프트웨어 유지보수와 재사용을 위한 재공학 Refactoring 기법 연구

박진호○ 이종호* 류성열

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

A Study of Re-Engineering Refactoring Technique for the Software Maintenance and Reuse

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

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

요 약

소프트웨어 유지보수는 노력과 시간이 많이 소요된다. 유지보수 되는 기존의 시스템들은 규모가 크고 복잡하기 때문에 운영비용의 대부분이 소프트웨어 유지보수에 소요되는 실정이다. 소프트웨어의 재사용은 이러한 문제점들을 해결하기 위하여, 소프트웨어 역공학, 재공학, 컴포넌트와 같은 많은 기법들이 제안되었다. 이런 기법들 중에서 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. 본 논문에서는 시스템의 유지보수와 재사용을 위한 재공학 Refactoring 기법을 제시한다.

1. 서 론

국제적 인터넷의 확산, 국가의 정책적인 컴퓨터 보급과 교육이 확대되면서 거대해진 경제시장의 규모에 걸 맞는 네트워크와 시스템들에 대한 새로운 기능과 더욱 빠른 속도를 요구하는 사용자들이 늘고있다. 이러한 기대에 부응하기위한 시스템의 교체 작업 들이 진행되고 있긴 하지만 너무 오래되고, 거대해진 시스템들을 교체하는 일은 비용과 노력면에서 많은 어려움이 있다. 이런 문제들을 해결하기 위한 방법으로 소프트웨어 재사용의 중요성 크게 부각되고 있다. 그러나 기존 시스템의 노후와 방대한 소프트웨어, 관련 문서들의 분실 등의 이유로 날로 늘어가는 사용자들의 요구를 만족시키지 못했다. 이러한 문제점들을 해결하기 위하여 소프트웨어의 유지보수와 재사용을 하는데 있어서 정확한 분석 작업을 필요로 하게 되었다. [1]

소프트웨어 재공학 기법 중에서 Refactoring 기법을 사용하면 소프트웨어 설계의 향상, 그리고 좀더 쉽고 이해하기 좋은 소프트웨어를 만들고, 버그(bug)를 좀더 쉽게 찾을 수 있도록 도와준다. 또한 새로운 기능을 추가하거나 코드를 다시 살펴보려고 할 때에도 많은 도움을 줄 것이다.

본 논문은 이런 사용자의 요구에 부응하기 위하여 UML을 응용하여 사용자가 좀더 쉽게 이해할 수 있도록 도와주는 Refactoring 다이어그램을 제시하고, 시스템을 유지보수하기 편리하게 기존의 시스템을 관리하고, 재사용 할 수 있는 PI-Refactoring 기법을 제시한다.

2. 관련 연구

2.1 객체지향 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 [2, 3, 4].

재공학의 대표적인 기법에는 재구성(Restructuring), 데이터 재공학(Data Reengineering), Refactoring 등이 있다. 재구성에는 문서/코드/데이터 재구성 등이 있다[5].

2.2 객체지향 소프트웨어 Refactoring

Refactoring 이란 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[6].

- Serge Demeyer 와 Stephane Ducassse 의 Refactoring 방법을 보면 클래스(Class) 다이어그램(Diagram)을 중심으로 한 5 단계 기법으로 다음과 같다[5].
  - 1 단계: Create Subclass
  - 2 단계: Move Attribute
  - 3 단계: Move Method

- 4 단계: Split Method + Move Method
- 5 단계: Clean-up

이 기법의 장점은 미리 서브클래스를 생성 후 요소들을 자리를 이동 시키기 때문에 중복되거나 그냥 지나칠 수 있는 요소들을 찾기가 쉽다. 즉, 요소들의 목록을 만들어서 관리하거나 바꾸기가 용이하다. 단점은 미리 만들어 놓고서 시작을 하기 때문에 꼭 필요한 요소들을 이동 시키는 오류를 범하거나, 다른 기능의 요소들이지만 다형성의 정의 없이 중복되게 덮어쓰는 경우들이 생긴다.

- Martin Fowler 가 주장하는 Refactoring 방법도 클래스 다이어그램을 중심으로 한 4 단계이고, 작업의 수행은 다음과 같다[6].

- 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

이 기법의 장점은 요소들을 추출하고, 이동하면서 적절한 사용 위치를 바꾸기 때문에 간단한 자리이동으로 작업을 끝낼 수 있다. 단점은 반복적인 자리이동이 생겨서 최적화된 성능을 발휘할 수 있는 요소들까지 자리이동이 이루어 진다는 것이다.

3. PI-Refactoring 기법

PI-Refactoring 이란 객체지향 소프트웨어의 Refactoring 기법 중에서 핵심적 기술이 되는 다형성과 상속성 (Polymorphism & Inheritance)을 이용하여 사용자를 위하여 쉽게 사용하고, 이해할 수 있도록 새롭게 정리한 객체지향 Refactoring 기법을 말한다.

본 절에서는 앞 절에서 설명한 객체지향적 Refactoring 방법들의 장점들을 통합하고, 단점으로 지적되는 과정을 생략하여 일반적이고도 표준적인 방법을 제시한다.

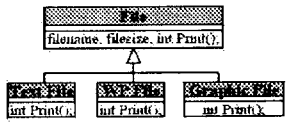

그림 1. 다중적으로 정의된 Print() 함수(Polymorphism)

그림 1 은 PI-Refactoring 4 단계 기법의 중요 개념인 다형성과 상속성을 이용하여 객체지향의 개념으로 각각의 요소들을 상황에 맞게 재정의 하고 상속하는 단계를 개념적으로 설명한 것이다[7].

본 절에서는 시스템의 구조적인 문제인 문서의 부족, 기능의 중복, 모듈방식의 부족, 부적당한 계층 등을 재공학 측면의 기법인 Refactoring 기법을 사용하여 해결책을 제시한다.

PI-Refactoring 의 첫번째 단계는 메소드(Method)/속성(Attribute) 추출 단계이고, 두 번째 단계는 메소드/속성 이동 단계이고, 세 번째 단계는 함수 분할 단계이고, 마지막으로 네 번째 단계는 분할 이동시킨 속성과 메소드들을 자연스럽게 만들어주는 최적화 단계이다. 그림 2 는 PI-Refactoring 의 4 단계를 나타낸다.

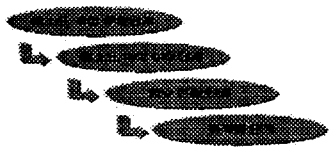

그림 2. PI-Refactoring 기법 4 단계

- **1 단계 – 메소드/속성 추출단계**
  : 어떠한 함수 내에 불필요한 메소드와 속성들을 찾아내는 단계이다.
- **2 단계 – 메소드/속성 이동단계**
  : 1 단계에서 찾아진 메소드와 함수들을 다른 클래스로 옮기는 단계이며, 이 메소드와 속성들이 더 많이 사용되는 함수를 찾아 기능별로 이동을 시킨다. 그 속성들이 가지고 있는 특징을 더 많이 사용하는 함수로 옮기고 이전 함수에서 호출하는 방법이다.
- **3 단계 – 함수 분할 단계**
  : 쿼리(Query)의 임시 저장소(Temp)와 같은 곳이 있으면 새로이 메소드와 속성들을 새로이 재배치 하여 하나의 독립된 함수로 분할하는 단계이다.
- **4 단계 – 최적화 단계**
  : 분할 되어진 함수의 메소드와 속성들을 자연스럽게 함수별로 정리하고, 수정하는 단계이다.

4. PI-Refactoring 다이어그램 모델링 기법

본 절에서는 앞 절에서 제시한 규칙들을 사용자가 좀 더 쉽게 이해하고, 사용할 수 있도록 도와주기 위해서 UML 의 클래스 다이어그램(=Cla D.)과 순차(Sequence) 다이어그램(=Seq D.)을 합하여 만든 Refactoring 다이어그램(=Refact D.)을 표 1 에서 제시한다[8].

표 1. PI-Refactoring 다이어그램

Refact D. = Cla D. + Seq D.

Refact D.을 간단하게 설명을 하면 그림 3 과 같다.

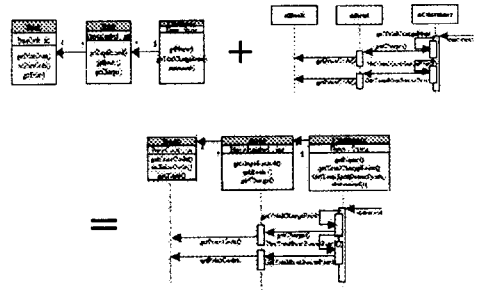

그림 3. PI-Refactoring 다이어그램 모델

- Refact D.의 장점은
  - 사용자가 분석된 자료를 가지고 클래스 다이어그램과 시퀀스 다이어그램을 중복하여 만들거나 찾는 번거로움을 줄일 수 있다는 것이다.
  - 또 하나는 Refactoring 작업에 효율성을 높일 수 있는 매우 간단하고 쉬운 프로세스로 이해와 효율을 증가시킬 수 있다는 것이다.

5. 사례 연구

본 절에서는 앞에서 제시한 Refactoring 기법을 이용하여 책 대여점의 적용과정을 보여준다.

- **책 대여점의 중요한 함수들의 기능은 다음과 같다.**
  - Book : 책 제목, 가격 코드 등의 전반적인 정보를 가지고 있는 함수이다.
  - Rent : 대여일수, 대여책 등의 정보 중 대여에 관련된 정보를 가지고 있는 함수이다.
  - Customer : 고객의 정보, 대여한 책 등에 대한 정보를 가지고 있는 함수이며, 상태(statement)를 가지고 대여한 책의

정보를 알 수 있다.

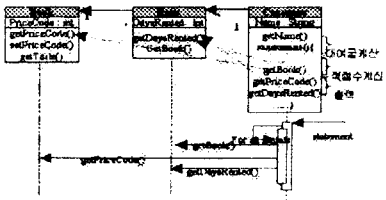

그림 4. 책 대여점 함수 분석

- **1 단계 – 메소드/속성 추출단계**
  : 함수 내에 불필요한 메소드와 속성들을 찾아내는 단계로 함수의 길이가 너무 긴 Statement()가 있고, Customer 함수가 Rent에서 Book에 이르는 함수를 호출하고 있다. 그러므로 추가적인 요구사항의 반영이 어렵고, 자료코드의 편중이 심해진다. 또한 코드 자체의 이해가 어려워진다.

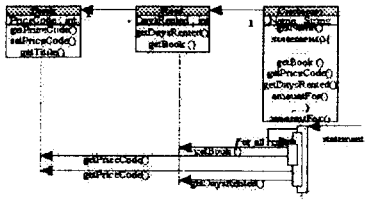

그림 5. 메소드/속성 추출 Refact D.

- **2 단계 – 메소드/속성 이동단계**
  : 1 단계에서 찾아진 메소드와 함수들을 다른 클래스로 옮기는 단계이며, 이 메소드와 속성들이 더 많이 사용되는 함수를 찾아 기능별로 이동을 시킨다. 여기에서 있는 Customer의 amountFor() 함수를 Rent의 getCharge()로 옮긴다. 즉, 앞에서 말한 속성들이 가지고 있는 특징을 더 많이 사용하는 함수로 옮기고 이전 함수에서 호출하는 방법이다. 이 단계에서 Customer의 대여비용 산출부분이 Rent 부분으로 넘어갔다. 따라서 Customer의 편중되었던 문제를 부분적으로 해결 할 수 있다. 하지만 아직도 Customer의 크기가 크고 Book에 이르는 함수들을 여전히 호출하고 있다.

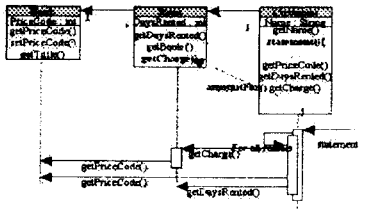

그림 6. 메소드/속성 이동 Refact D.

- **3 단계 – 함수 분할 단계**
  : 2 단계 작업 후 시스템 성능의 향상을 위해 독립적인 함수로 분할하는 단계로, 메소드와 속성들을 새로이 재배치 하여 하나의 독립된 함수로 분할한다. 이 단계에서는 대여 보너스 점수 계산에 대한 Extract와 Book 메소드를 적용한다. 따라서 Customer에서 Book의 멤버를 호출하는 부분들의 문제는 해결이 되었다. 하지만 아직 Statement() 함수의 세분화가 더욱 필요하다. 예를 들어 Statement()함수를 getTotalChargePoint()와 getTotalRentBonusPoint()로 나눈다.

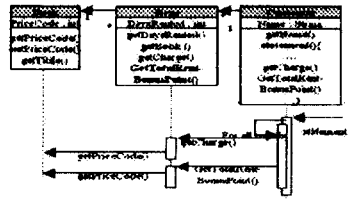

그림 7. 함수 분할 Refact D.

- **4 단계 – 최적화 단계**
  : 분할 되어진 함수의 메소드와 속성들을 자연스럽게 함수별로 정리하고, 수정하는 단계이다.

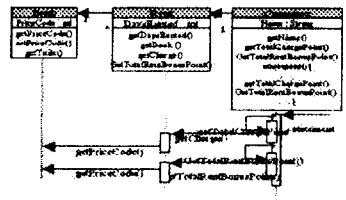

그림 8. 최적화 Refact D.

6. 결론 및 향후 연구과제

객체지향 재공학 중에서도 Refactoring의 중요성은 날로 커지고 있다. 본 논문은 핵심적인 사항들을 정리하여 PI-Refactoring 기법과 모델을 제시하였다. 새로이 제시한 방법을 이용하여 소프트웨어 유지보수와 재사용을 위해 좀더 정확하고 경제적인 방법으로 Refactoring하는 기술을 더욱 발전시킬 수 있을 것이다. 아직 다양한 기능들을 전문적으로 지원하는 시스템들을 모두 만족 시킬 수는 없고, 전반적인 과정에 대한 소개가 이루어졌지만 앞으로는 더욱 전문적이고 기술적인 기법이 개발되어야 할 것이다. 프로그램의 구현에 있어서도 기술적인 부분적 도입이 가능하므로, 프로토타입(Prototype) 모델을 사용하여 실생활에 적용할 수 있는 프로그램의 구현도 멀지 않아 가능해질 것이다. 특히, 재사용의 관심이 높아지고 있기 때문에 기능 단위의 분할이나 이동 등을 좀더 쉽게 할 수 있는 객체지향, 컴포넌트화 방법이나 자동화 도구의 개발 분야의 연구가 더욱 진행되어야 할 것이다.

7. 참 고 문 헌

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

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

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

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

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

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

[7] 김수동, "실무자를 위한 소프트웨어공학", 에드텍, 1999.

[8] 박진호,김수동,류성열,"UML 다이어그램들 간의 일관성 검증 방법",한국정보과학회,춘계학술발표논문집 P524~526, 1998.