ISSN 2234-4438
중소기업융합학회 논문지 제6권 제4호 pp. 137-150, 2016 DOI : http://dx.doi.org/10.22156/CS4SMB.2016.6.4.137

소스 코드 품질 향상을 위한 리팩토링 기법 및 도구 분석

김두환, 정유진, 홍장의*
충북대학교 소프트웨어학과

Analysis of Refactoring Techniques and Tools for Source Code Quality Improvement

Doohwan Kim, YooJin Jung, Jang-Eui Hong*
Department of Computer Science, Chungbuk National University

요 약 IT 기술 및 비즈니스의 급속한 발전에 따라 새로운 서비스를 고객에게 제공하기 위한 노력이 증가하고 있으며, 신속한 서비스 제공을 위해 기존의 레거시 시스템에 대한 개선 및 확장이 빈번히 발생하고 있다. 이로 인하여 기존 레거시 시스템에 대한 소스 코드의 품질 확보는 서비스 요구에 신속히 대응할 수 있는 핵심적인 기술 요소가 되었다. 리팩토링은 기존 레거시 코드에 대한 품질을 확보하기 위한 공학적 기술로서, 부가가치를 제공하는 레거시 시스템의 개선 및 확장에 필수적이다. 본 논문에서는 레거시 시스템의 소스 코드 품질 향상을 위한 기존의 리팩토링 기법과 지원 도구에 대한 조사 분석을 통해 리팩토링 기법과 도구에 대한 특성을 제안한다. 제안하는 특성을 기반으로 서비스 개발자가 레거시 시스템의 소스 코드 품질 향상을 위하여 리팩토링을 수행하고자 하는 경우, 어떠한 기법과 도구를 활용할 것인가에 대한 가이드라인을 제공받을 수 있다. 이를 통해 보다 정확하고 시행착오 없는 레거시 시스템의 소스 코드 품질을 향상시킬 수 있으며, 새로운 서비스에 대한 신속한 대응도 가능하게 될 것이다.

키워드 : 소스 코드 품질, 코드 리팩토링, 리팩토링 기법, CASE 도구

Abstract Along with the rapid development of IT technology and business services, the effort to provide new services to the customers has been increasing, and also the improvement and enhancement of legacy systems are continuously occurring for rapid service delivery. In this situation, the quality assurance of the source code for the legacy system became a key technical elements that can quickly respond to the service needs. Refactoring is an engineering technique to ensure the quality for the legacy code, and essential for the improvement and extension of the legacy system in order to provide value-added services. This paper proposes some features of refactoring techniques through surveying and analyzing the existing refactoring techniques and tools to enhance source code quality. When service developers want to refactor the source code of the legacy system to enhance code quality, our proposed features may provide with the guidance on what to use any technique and tool in their work. This can improve the source code quality with correct refactoring and without trial and error, and will also enable rapid response to new services.

Key Words : Source code quality, Code refactoring, Refactoring technique, CASE tool

1. 서 론

소프트웨어는 현대 사회에서 필수 불가결한 핵심 기반 요소가 되었으며, 정보기술의 신속한 발전과 삶의 질에 대한 급속한 변화에 따라 소프트웨어의 응용이 더욱 증가하고 있다. 또한 이러한 변화로 인하여 사용자에게

Received 2016-10-30 Revised 2016-11-15 Accepted 2016-11-28 Published 2016-12-31
*Corresponding author : Jang-Eui Hong (jehong@chungbuk.ac.kr)

제공하는 서비스의 수명주기는 더욱 짧아지고 있으며, 서비스 개발자는 이에 대응하기 위한 서비스 소프트웨어 개발에 많은 노력을 기울이고 있다.

기존의 레거시 시스템을 통한 서비스 환경에서 새로운 서비스의 개발에 대한 요구는 기존 시스템에 대한 빈번한 개선과 기능 확장을 필요로 하고 있다[1]. 이러한 개선과 확장은 레거시 시스템의 소스 코드에 대한 지속적인 변경을 수반한다[2, 3] 따라서 보다 신속하고 정확한 서비스 개발을 위하여 레거시 시스템의 소스 코드에 대한 품질 확보가 무엇보다도 중요한 서비스 개발의 핵심 요소가 되었다. 레거시 시스템의 높은 소스 코드 품질은 개발자가 정확하고 오류 없이 새로운 기능을 레거시 시스템에 추가하기 위한 필수 조건이며, 이를 통해 보다 안정적인 서비스 제공이 가능하게 된다.

기존 레거시 시스템에 대한 소스 코드의 품질을 확보하기 위한 공학적 방법의 하나가 리팩토링 기술[4]이다. 리팩토링은 기존 사용자 서비스의 기능에 대한 변화 없이 소스 코드를 재구조화 하는 활동으로서, 소스 코드에 대한 가독성 및 이해성을 높이고, 새로운 변경을 용이하게 수행하기 위한 모듈화 구조의 소스 코드를 생성하는데 그 의의가 있다[5, 6]

과거 리팩토링과 관련된 연구들이 많이 제시되어 왔다. 특히 M. Fowler는 소스 코드가 갖는 Code Smells을 정의하고, 이러한 Smells을 제거하기 위한 68여개의 소스 코드 리팩토링 기법을 제안하였다[7]. J. Garcia는 Fowler의 기법을 확장하여 성능을 개선하기 위한 리팩토링 기법을 제안하였다[8]. 또한 A. Verto와 J. Lee의 연구에서는 소스 코드의 소모 전력을 절감하기 위한 리팩토링 기법을 제안하였다[9-12].

또한 제안된 리팩토링 기법을 지원하기 위한 다양한 도구들도 개발되어 왔다. Eclipse 재단에서 개발한 Eclipse Refactor는 27가지의 코드 리팩토링 기법을 지원하는 자바 중심의 리팩토링 도구이며, Telerik 사에서 개발한 JustCode는 C#언어에 대하여 32가지 리팩토링 기법을 지원한다[13-14]. 그 외에도 Whole Tomato 사의 Visual Assist, JetBrains 사의 ReSharper C++와 같은 리팩토링 도구들이 개발되어 사용하고 있다[15-16].

과거 다양한 리팩토링 기법과 이를 지원하는 도구들이 개발되어 왔지만, 레거시 시스템에 대한 소스 코드의 품질을 지속적으로 확보해야 하는 서비스 개발자들은 다음과 같은 문제들로 아직도 고민하고 있다.

- 기존 레거시 코드에 대한 재구조화를 시도하기 위하여 어떤 부분을 변경해야 할 것인지 쉽게 결정하기 어렵다.
- 변경을 위한 코드를 식별한 경우에도 어떤 기법을 적용하여 재구조화해야 하는지 리팩토링 기법을 선택하기에 충분한 이해가 부족하다.
- 리팩토링을 지원하기 위한 다양한 도구로부터 적합한 도구를 선택하기 어렵다
- 리팩토링을 수행한 후에, 코드의 가독성 증진은 물론, 성능이나 소모전력 절감과 같은 품질 요소도 개선하기를 원한다.

이와 같은 어려운 점을 해결하기 위한 목적으로 본 논문에서는 기존의 제안된 리팩토링 기법과 지원 도구들에 대한 조사 분석을 수행하였다. 다양한 기법과 도구를 체계적으로 분류하고, 또한 리팩토링 도구가 가져야 할 특성(feature)를 정의하여 리팩토링 도구의 특성을 체계적으로 정리한다. 제안하는 리팩토링 도구의 특성은 임의의 특성을 지원하는 기법과 도구가 무엇인지를 쉽게 식별할 수 있는 가이드라인으로 활용할 수 있으며, 이를 통해 보다 정확 신속하고, 또한 시행착오 없이 리팩토링을 수행할 수 있도록 지원한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 리팩토링 기법에 대하여 조사 분석하였으며, 3장에서는 리팩토링 지원 도구에 대하여 분석하였다. 또한 4장에서는 소스 코드 품질 특성의 하나로 소모 전력 절감 효과를 제공하는 리팩토링 기법들에 대하여 조사 하였다. 5장에서는 이러한 기존 연구의 분석 결과를 기반으로 리팩토링 특성을 정리하였으며, 6장에서는 도구 선정을 위한 가이드라인을 제안하였다. 마지막으로 7장에서는 결론 및 향후 연구에 대하여 제시하였다.

2. 리팩토링 기법 분석

2.1 리팩토링 기법의 역사

코드에 대한 재구조화가 오랫동안 진행되어 왔지만, 코드 리팩토링에 대한 연구는 1990년 W. Opdyke와 R. Johnson의 연구 논문에서 처음 제시되었다[17]. 그 이후 1991년 W. Griswold의 박사 학위 논문에서 리팩토링 기법에 대한 구체적인 연구결과가 제시되었고, 1992년 W. Opdyker의 박사 학위논문에서 객체지향 패러다임을 근

간으로 하는 리팩토링 기법이 제시되었다[18-19].

그러나 코드 기반의 리팩토링 기법에 대한 가장 체계화된 연구는 2002년에 발간된 M. Fowler의 'Refactoring: Improving the Design of Existing Code' 에서 출발한다[7]. M. Fowler는 소스 코드에서 탐지될 수 있는 문제 발생 요인들을 Code Smells로 정의하고, 이들을 제거하기 위한 리팩토링 기법을 체계화하였다.

2.2 리팩토링 기법의 분류 기준

2.2.1 품질 특성에 의한 분류

리팩토링 기법에 대한 연구는 매우 다양한 목적을 달성하기 위해 제시되었다. 이들 연구는 소스 코드에 대한 리팩토링을 통해서 얻고자 하는 효과가 무엇인가에 따라 분류할 수 있다. 리팩토링 기법의 적용 목적은 그 간의 연구들을 종합할 때, 다음과 같은 3가지 목적으로 정리할 수 있다.

(1) 유지 보수성 : 복잡하게 엉킨 소스 코드를 재구조화하면서, 코드의 모듈화 특성을 향상시키는데 그 목적이 있다. 즉, 코드에 대한 이해성을 높여서 유지보수성을 향상시키고자 하는 것이다.
(2) 성능 : 소스 코드의 재구조화를 통해 코드에 대한 이해성을 높이는 것과 동시에 성능의 개선을 추구하는 연구들도 진행되었다.
(3) 소모전력 : 임베디드 시스템이나 모바일 소프트웨어의 경우, 소모전력 절감의 품질 특성을 요구한다. 따라서 리팩토링을 통하여 소모 전력을 절감하고자 하는 연구들이 최근에 진행되었다.

이와 같은 소프트웨어의 품질 특성을 기반으로 하여 수행된 기존의 연구들을 분류하면 표 1과 같이 정리할 수 있다. Table 1에서 보는 바와 같이 M. Fowler 연구는 소스 코드의 유지보수성 향상에 주안점을 두었으며, J. Garcia의 연구는 성능 향상에, 그리고 A. Verto의 연구는 소모전력 절감에 리팩토링 기법 연구의 목적이 있었다.

이들의 연구에서는 단일 리팩토링 기법만을 제시한 것이 아니라 매우 다양한 측면에서의 코드 재구조화 방법을 제시하였으며, 각 방법은 명칭, 사용 의도, 적용 방법, 그리고 적용 예제 등을 통해 명세하고 있다.

Table 1. Classifying Refactoring Techniques by Quality Factor

Authors	Proposed Techniques	Quality
M. Fowler [7]	•Duplicated Code •Long Method •Large Class •Long Parameter List •Data Clump •Temporary Field •Incomplete Library Class •Message Chains, etc.	Maintain-ability
J. Garcia [8]	•Removing Globals •Sequences and Structures •Parallel Library •Strategy Encapsulation •Avoiding Data Race, etc.	Performance
A. Verto [9]	•Parameter By Value •Self Assign •Mutual Exclusion OR •Switch Redundant Assign •Dead Local Store •Non Short Circuit, etc.	Energy Consumption
J. Lee [10]	•Complex Expressions •Common Sub-expressions •Tail Recursion •Loop Structure •Dead Code	

2.2.2 기법의 패턴에 의한 분류

리팩토링 기법이 어떤 패턴에 의해 적용되는 가에 따라 다양한 기법이 제시되었다. 그중에서도 M. Fowler가 제시한 리팩토링 기법의 분류는 각 기법의 적용 방법에 대한 패턴을 유형화하였다. 이러한 적용 절차상에서 나타나는 패턴의 유형에 따라 기존 연구들을 분류하면 Table 2와 같다. Table 2에서 제시하는 리팩토링 기법의 적용 패턴의 의미를 설명하면 다음과 같다.

(1) Abstraction : 정보의 은닉과 일반화 개념의 적용을 통하여 코드 접근 및 공유에 대한 안전성을 제공하기 위한 리팩토링 패턴
(2) Breaking : 기존 코드를 논리적인 더 작은 코드 조각으로 분리함으로서, 변경의 영향 및 재사용성을 증진하거나, 또는 하나의 Class 또는 Method를 쪼개서 새로운 클래스를 생성하기 위한 리팩토링 패턴
(3) Moving : 현재 위치의 코드 조각을 다른 Method나 Class(혹은 모듈)로 이동함으로서, 객체간의 상호작용을 줄이고 이를 통한 변경 영향을 최소화하기 위한 리팩토링 패턴
(4) Substitution : Method나 Class의 이름을 변경함으로서, 코드에 대한 가독성을 높이고 유지보수를 쉽게 하기 위한 리팩토링 패턴.

Table 2. Classifying Refactoring Techniques by Applying Pattern

Patterns	Technique Styles	Authors
Abstraction	• Encapsulation • Generalize type	M.Fowler [7] J. Garcia [8]
Breaking	• Componentization • Extract class • Extract method	M.Fowler [7] A. Verto [9] J. Lee [10]
Moving	• Moving method/field • Rename method/field • Pull up • Pull down	M.Fowler [7] A. Verto [9] J. Lee [10]
Substitution	• Replace Type-checking • Replace condition	M.Fowler [7] J. Lee [10]

2.2.3 적용 언어에 의한 분류

리팩토링 기법이 어떤 프로그래밍 언어를 대상으로 하는 가에 따라서도 기존의 연구들을 분류할 수 있다. 초기의 코드 재구조화에서는 시스템 프로그래밍에 대한 어셈블리어 수준에서의 재구조화도 진행되었지만, 이들에 대한 분석은 진행하지 않았다[26, 27]. Table 3은 리팩토링 기법이 적용되는 대상 언어를 기준으로 기존 기법 및 연구들을 분류한 것이다.

Table 3. Classifying Refactoring Techniques by Target Language

Languages	Representative Techniques	Authors
C	•Encapsulate Field •Introduce Explaining Var.	J. Lee[10] A. Garrido [20]
C++	•Extract Class/Method •Rename method	M. Fowler [7] A. Verto [9]
C#	•Extract Method •Pull up Field/Method	L. Hunt [21] M. Gatrell [22]
Objective -C	•Encapsulate Field •Push Down Field/Method	Apple Dev. [23]
Java	•Extract Interface/Field •Rename Method	M. Schafer [24] D. Dig [25]

2.3 M. Folwer의 리팩토링 기법

기존의 리팩토링 연구에서 가장 널리 알려진 M. Fowler의 리팩토링 기법에 대하여 구체적으로 살펴보면 Table 4와 같이 68개의 기법으로 정리된다.

Table 4. Refactoring Techniques by M. Fowler [7]

Category	No.	Code Refactoring Techniques
Composing Method	R1	Extract Method
	R2	Inline Method
	R3	Inline Temp
	R4	Replace Temp with Query
	R5	Introduce Explaining Variable
	R6	Split Temporary Variable
	R7	Remove Assignments to Parameters
	R8	Replace Method with Method Object
	R9	Substitute Algorithm
Moving Features Between Objects	R10	Move Method
	R11	Move Field
	R12	Extract Class
	R13	Inline Class
	R14	Hide Delegate
	R15	Remove Middle Man
	R16	Introduce Foreign Method
	R17	Introduce Local Extension
Organizing Data	R18	Self Encapsulate Field
	R19	Replace Data Value with Object
	R20	Change Value to Reference
	R21	Change Reference to Value
	R22	Replace Array with Object
	R23	Duplicate Observed Data
	R24	Replace Magic Number with Symbolic Constant
	R25	Encapsulate Field
	R26	Encapsulate Collection
	R27	Replace Record with Data Class
	R28	Replace Type Code with Class
	R29	Replace Subclass with Fields
Simplifying Conditional Expression	R30	Decompose Conditional
	R31	Consolidate Conditional Expression
	R32	Consolidate Duplicate Conditional Fragments
	R33	Remove Control Flag
	R34	Replace Nested Conditional w/ Guard Clauses
	R35	Replace Conditional with Polymorphism
	R36	Introduce Null Object
	R37	Introduce Assertion
Making Method Calls Simpler	R38	Rename Method
	R39	Add Parameter
	R40	Remove Parameter
	R41	Separate Query from Modifier
	R42	Parameterize Method
	R43	Replace Parameter with Explicit Methods
	R44	Preserve Whole Object
	R45	Replace Parameter with Method
	R46	Introduce Parameter Object
	R47	Remove Setting Method
	R48	Hide Method
	R49	Replace Constructor with Factory Method
	R50	Replace Error Code with Exception
	R51	Replace Exception with Test
Dealing with Generali-zation	R52	Pull Up Field
	R53	Pull Up Method
	R54	Pull Up Constructor Body
	R55	Push Down Method
	R56	Push Down Field
	R57	Extract Subclass
	R58	Extract Superclass
	R59	Extract Interface
	R60	Collapse Hierarchy
	:	:
	R67	Change Unidirectional Asso. to Bidirectional
	R68	Change Unidirec. Association to Bidirectional

3. 리팩토링 도구 분석

3.1 리팩토링 도구의 공통 특징

3.1.1 개발환경과의 종속성

일반적으로 리팩토링은 작성중이거나 이미 작성된 코드를 대상으로 한다. 따라서 리팩토링은 소프트웨어 개발 단계 중 구현 단계와 밀접한 관계를 갖는다. 이러한 관점에서 리팩토링을 지원하기 위한 도구들은 대부분 Fig. 1에서 보이는 바와 같이 구현 도구에 포함되어 있거나 플러그인 형태로 동작하는 특징을 갖는다.

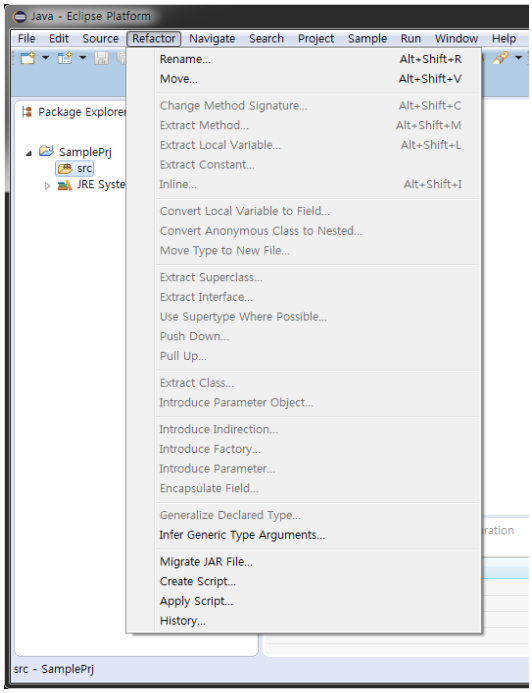


Fig. 1. The Refactoring Menu in Eclipse

Fig. 1과 같이 리팩토링 기능은 Eclipse에서 제공하는 기본적인 기능들을 포함하고 있으며, 다양한 Eclipse Plug-in들을 통해 해당 기능에서 기존에 지원되지 않았던 기법들을 포함하도록 확장할 수 있다.

하지만 앞서 언급한 것과 같이 Eclipse 재단에서 수행되고 있는 프로젝트나 플러그인을 사용한다고 하더라도 대부분의 경우는 Eclipse에서 지원되는 기본 기능에 대한 확장의 형태로 동작되도록 설계되어 있다. 이러한 특징은 소프트웨어 구현을 지원하기 위한 IDE(Integrated Development Environment)와 분리되었을 때, 작성한 코드를 리팩토링하기 위한 도구에서 다시 불러오고 분석해야 하는 번거로움을 방지하고 IDE와의 기능 중복 등을 피할 수 있다는 장점에서 기인한다.

3.1.2 반자동화된 리팩토링 지원

리팩토링을 자동화된 기능으로 지원하기 위한 도구가 제작되어 있더라도, 실제 코드를 분석하여 리팩토링해야 하는 지점을 찾는다는 것은 매우 어려운 일이다. 때문에 리팩토링 기법을 지원하는 방법에 있어서 대부분의 도구들이 반자동화되어 있다는 특징이 있다.

예를 들어, 사용자가 Extract Method를 수행하고자 할 때, 코드 내에서 반복되고 있는 부분을 블록으로 지정한 후 리팩토링 도구의 메뉴에서 Extract Method를 선택하면 리팩토링을 지원하기 위한 팝업창을 통해 새로 생성되는 Method의 이름, 반환형 및 인자의 목록 등을 입력하게 된다.

해당 기능의 수행 결과로써 당초 사용자가 블록으로 지정한 부분은 Method 호출로 변경되며, Class 내부에 새로운 Method가 생성된다. 이처럼 대부분의 리팩토링 기법들은 반자동화된 기능을 통해 제공된다. 따라서 리팩토링을 수행하고자 하는 소프트웨어 엔지니어는 도구를 이용하더라도 리팩토링 기법의 종류, 이름, 용도 및 적용 방법 등에 대한 배경지식이 있어야만 이를 활용할 수 있다.

3.1.3 리팩토링 적용 프리뷰

리팩토링 기법은 기본적으로 작성된 코드의 기능성은 변화시키지 않으면서 구조변경을 통해 품질을 향상시키는 것에 그 목적이 있다. 따라서 Fig. 2와 같이 코드의 변경에 있어서 어떠한 부분이 어떻게 변경되었는지에 대한 정보를 사용자에게 확인할 수 있도록 하는 기능이 포함된다.

이는 사용자의 의도와 부합하는 코드 변경이 이루어졌는지를 확인할 수 있도록 하는 동시에 변경되는 부분을 식별할 수 있게 하는 효과가 있다. 하지만 해당 기능은 단순한 변수 또는 함수(Method) 등의 이름을 변경할 때는 사용되지 않는 경우도 있다. 다만 리팩토링 도구에 따라서는 변경되는 부분을 추적하면서 사용자에게 적용 여부를 확인받기도 한다.

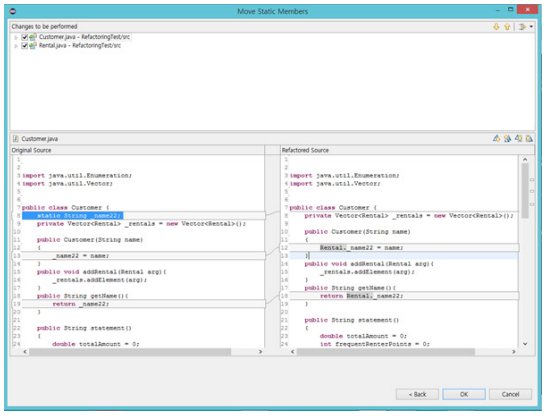


Fig. 2. Code Comparing Function for Refactoring

3.1.4 한정된 프로그래밍 언어 지원

리팩토링은 작성된 코드의 구조를 변경하기 때문에 특정 언어에 한정적으로 지원이 가능한 경우가 많다. C와 C++의 경우 구조적인 유사성을 가지고 있지만 각각이 포함하고 있는 언어의 패러다임(구조적, 객체지향적)이 다르기 때문에 다소 차이를 나타낼 수 있으며, C와 JAVA의 비교 관점에서는 구조적 유사성이 매우 적어지게 된다.

또한 리팩토링 기법 중 일부는 프로그래밍언어의 패러다임과 종속적인 특성을 가지고 있기도 하다. 따라서 다양한 언어를 지원하는 것에는 한계가 있으며, 그로 인하여 각 리팩토링 도구가 대상으로 하는 언어도 한정되게 된다.

3.2 대표적인 리팩토링 도구들

현재 다양한 상용 또는 오픈소스 기반의 리팩토링 지원 도구가 개발되어 사용되고 있다. 그 중 대표적인 리팩토링 도구(IDE 통합 형태 포함)들을 다음과 같다.

(1) Eclipse CDT [13]
(2) Eclipse JDT [28]
(3) Visual Assist X [15]
(4) ReSharper C++[16]
(5) Refactor! for C++ [29]
(6) Xcode [30]
(7) JustCode [14]
(8) NetBeans [31]

먼저 Eclipse의 경우 기본적으로 개발환경 자체에서 제공하는 리팩토링 기능이 존재한다[13, 28]. 해당 기능은 매우 기초적인 리팩토링 기법은 물론이고 Extract Method와 같은 기능들 또한 제공 된다. Eclipse는 기본적으로 JAVA 기반의 개발환경 JDT(JAVA Development Tool)로 이용되었으나, 최근 GCC 등과 연동하여 C언어 개발을 지원하는 CDT(C/C++ Development Tool) 환경을 지원하고 있다. 또한 Eclipse가 JAVA 기반으로 개발된 도구이기 때문에 Windows는 물론이고 Linux나 MAC OS와 같은 다양한 플랫폼에서 동작할 수 있다는 특징을 가지고 있으며, Eclipse Plug-in을 통해 리팩토링 기능을 확장 및 보완 할 수 있다[32-35].

Visual Assist는 Microsoft의 Visual Studio에서 사용되는 Plug-in으로 동작하며, M. Fowler가 제안한 68가지 리팩토링 기법 중 5가지를 지원하며, 그 외에도 다양한 기본적인 리팩토링 기능을 지원하고 있다. ReSharper C++의 경우 Visual Assist와 같이 Visual Studio의 Plug-in 형태로 동작한다. M. Fowler의 리팩토링 기법 중 3가지를 지원하고 있다.

JustCode도 앞서 언급한 ReSharper 및 Visual Assist와 같이 Visual Studio상에서 사용되는 Plug-in이다. Xcode는 Apple에서 제공하는 개발환경으로서, MAC OS에서 동작한다. 때문에, Apple 플랫폼에서 사용되도록 개발된 언어를 중심적으로 지원하고 있으며, Plug-in을 통해 C# 등을 지원할 수 있다는 특징을 갖는다. JavaScript 및 HTML과 같은 언어 또한 지원하며, M. Flower의 기법 중 10가지를 지원하고 있다.

마지막으로 NetBeans의 경우는 Eclipse와 같이 JAVA 기반으로 개발된 개발도구이다. 마찬가지로 여러 가지 리팩토링 기능을 지원하지만, Eclipse보다는 다소 간소하게 구성되어 있다.

3.3 리팩토링 도구의 분류

리팩토링 도구들은 리팩토링 기법을 활용하여 소프트웨어 엔지니어가 작성한 코드의 품질향상을 위해 사용되기 때문에 지원언어, 지원 기법 및 IDE 환경 등의 다양한 요소로 분류가 가능하다. 이중 IDE 환경의 경우, 앞서 설명한 것과 같이 해당 리팩토링 도구의 동작 환경에 대한 내용이며, 본 논문의 3.2절에서 이에 대한 내용을 다루었다. 따라서 본 장에서는 해당 분류 기준을 제외한 지원 언어와 지원 기법을 중심으로 3.2절에서 제시한 리팩토

링 도구들에 대한 분류를 수행한다.

3.3.1 언어 중심 분류

앞서 서술한 바와 같은 리팩토링 도구는 기법 특성에 따른 지원 가능한 언어가 한정되어 있다. 이러한 특성으로 인해 리팩토링을 수행하고자 하는 대상 프로그램의 언어에 따라 활용 가능한 리팩토링 도구의 범위가 결정된다. 이에 따라 본 논문에서는 3.2절에서 소개한 리팩토링 도구들을 우선적으로 Table 5에서와 같이 지원 가능 언어에 따라 분류하였다.

Table 5에서, IDE마다 지원하는 언어의 범주가 한정되어있기 때문에 본 논문에서 분류 대상으로 선택한 언어를 모두 지원하는 리팩토링 도구는 존재하지 않았다. 하지만, 별도의 Plug-in을 설치한다면 지원 범위를 늘릴 수 있을 것이라고 생각된다. (1)과 (2)의 Eclipse는 기본적으로 같은 플랫폼을 공유하고 있으며, Eclipse JDT라고 하더라도, Plug-in 설치를 통해 CDT환경을 동시에 사용할 수 있다. 즉, 향후 확장 가능성을 고려한다면 Eclipse의 경우 그 지원 대상 언어가 더 많아질 수 있을 것이라고 판단한다.

다음으로 (3)에서 (6)의 경우 모두 Microsoft의 Visual studio 상에서 운영되는 Plug-in이기 때문에 Visual studio의 지원 언어에 종속될 수 밖에 없다. 하지만 이들 도구들 중에서도 Visual studio에서 주로 다루고 있는 언어인 C, C++, 그리고 C#을 모두 지원하는 도구는 (3)이 유일하다. 따라서 C언어 기반의 Windows 플랫폼 프로젝트를 폭넓게 사용한다면 지원언어의 관점에서 (3)이 가장 적합할 것이다. Xcode는 유일하게 Objective-C/C++을 지원하는 리팩토링 도구이다. 이는 Apple 플랫폼에서 동작하는 프로그램을 작성하기 위해 개발된 IDE이기 때문이며, 플러그인을 통해 C#을 지원하기도 하지만 윈도우 플랫폼에서 사용되는 GUI 환경까지 Xcode에서 작성하기에는 무리가 있으며, Apple 플랫폼에서 운영되는 C/C++ 기반의 프로젝트에 적합하다고 할 수 있다.

Netbeans의 경우 C와 C++ 및 JAVA를 지원하여 Eclipse와 유사한 언어 지원 범위를 보이고 있다. 따라서 Eclipse와 Netbeans 중에서 선택해야 한다면 3.3.2절에서 제시하는 기법 중심의 분류를 참조하여 의사결정을 수행할 수 있을 것이다.

3.3.2 기법 중심 분류

리팩토링 도구 선택에 대해서는 운영되는 환경과 언어가 우선적으로 고려되어야 하지만, 해당 항목들을 만족하는 도구가 여러 가지 있다면 물론 지원하는 기법에 따른 선택이 이루어져야 한다. 본 장에서는 앞서 제시한 8가지의 리팩토링 도구들이 어떠한 기법들을 지원하고 있는지의 관점에서 Table 6와 같이 비교를 수행한다.

Table 6에서 제시하는 리팩토링 기법 목록은 현재 도구가 지원하는 기법을 중심으로 나열한 것이다. 이들 기법은 M. Fowler가 제시하는 68개 기법에 모두 포함된다. 가장 많은 기법을 지원하는 것은 Eclipse이다. (1)과 (2)는 기본적으로 같은 플랫폼을 이용하기 때문에 그 지원 범위가 같다. 물론 CDT를 이용 할 때와 JDT를 이용할 때 가용 기법의 범위는 다소 달라질 수 있다. Eclipse 재단에서 개발 중인 프로젝트 등을 통해 해당 기능은 더욱 강화될 수 있을 것으로 생각된다.

MS Visual Studio 상에서 동작하도록 개발되어 있는 (3)에서 (6)의 경우 지원 플랫폼이 동일하고, 지원 언어가 유사하였다. 그 중 가장 많은 언어를 지원했던 것은 Visual Assist X였지만, 지원하는 기법의 관점에 대해서는 Just Code가 좀 더 많은 기법을 지원한다는 이점을 갖는다. 때문에 C/C++, C#을 기반으로 진행하는 프로젝트

Table 5. Refactoring Tool Classification by Supporting Language

IDE Tools	C	C++	C#	Objective-C	JAVA
(1) Eclipse CDT	●	●			
(2) Eclipse JDT					●
(3) Visual Assist X	●	●	●		
(4) ReSharper C++		●			
(5) Refactor! for C++	●	●			
(6) Xcode			●	●	
(7) JustCode	●	●	●		
(8) NetBeans	●	●			●

Table 6. Refactoring Tool Classification by Supporting Techniques

Techniques	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Add Parameter	●	●	●					●
Encapsulate Field	●	●	●		●		●	●
Extract Super Class	●	●						●
Extract Class	●	●						
Extract Interface	●	●				●		●
Extract Method	●	●	●	●	●	●	●	●
Inline Method	●	●				●		
Inline Temp	●	●				●		
Introduce Explaining Variable	●	●	●	●		●		
Introduce Parameter Object	●	●						
Move Field	●	●						
Move Method	●	●						
Pull Up Field	●	●				●	●	
Pull Up Method	●	●				●	●	
Push Down Field	●	●					●	
Push Down Method	●	●					●	
Remove Parameter	●	●				●		
Remove Setting Method					●			
Rename Method	●	●	●	●	●	●	●	●
Replace Constructor with Factory Method	●	●				●		
Replace Magic Number with Symbolic Constant	●	●						

가 많은 조직의 경우, 어떤 언어를 많이 사용하는가에 대한 빈도수에 따라 Visual Assist X나 JustCode 중 한 가지를 선택하는 것이 좋을 것이다. Xcode는 지원 기법의 개수에 대해서는 평균 수준을 갖추고 있으나, 본 논문의 비교 대상 중 유일하게 Apple 플랫폼을 대상으로 운영된다는 점에서 이점을 가지고 있다.

Netbeans는 지원 언어 및 운영 환경 등의 범위가 Eclise와 매우 유사하지만, 지원 가능한 리팩토링 기법의 수에서는 현저하게 적다는 점을 고려할 때, Netbeans보다는 Eclipse를 이용하는 것이 리팩토링 관점에서 다소 유리할 것으로 예상된다.

4. 에너지 기반 리팩토링 도구

4.1 필요성

3장에서 제시한 리팩토링 도구들은 모듈화 등의 품질 속성을 향상시키는 것에 주목적을 두고 있다. 하지만 코드의 구조변경으로부터 향상시킬 수 있는 품질 속성은 그 외에도 Performance, Energy Efficiency 등 다양하게 정의될 수 있다. 특히 최근 모바일 소프트웨어의 활용도가 급증하며, IoT(Internet of Things) 환경의 확장이 예상되고 있는 가운데 저전력 소모는 아주 중요한 품질 속성 중 하나로 인식되어 오고 있다. 이러한 관점에서 저전력 소모를 지원하는 리팩토링 기법들과 이를 이용할 수 있도록 하는 리팩토링 도구에 대한 필요성이 강조되고 있다[10].

저전력 소모를 위한 소프트웨어 개발 기술은 아키텍처 수준, 디자인 수준 및 코드 수준 등에서 수행되고 있으며 특히 코드 수준의 경우 전적으로 같은 행동을 하지만 다른 구성을 갖는 코드를 통해 전력 소모량에 대한 개선을 수행할 수 있다. 이는 리팩토링이 가지고 있는 기본적인 원칙과도 부합하고 있기 때문에, 저전력 소모를 위한 리팩토링 기법 및 도구는 향후 아주 유용한 개발지원 도구가 될 것으로 예측된다.

4.2 주요 리팩토링 기법

모바일 소프트웨어는 그것의 운영 전력을 전적으로 배터리에 의존하고 있다는 점에서 저전력 소모에 대한 수요를 증가시킨다. 저전력 소모를 위한 리팩토링 기법의 대표적인 연구들은 A. Vetro의 연구와 J. Lee의 연구가 있다[91 10]. 해당 연구들은 기존의 리팩토링 기법을 통해 개선해야 할 코드의 형태를 소모 전력에 초점을 맞추어 Energy Code Smells을 정의하고, 이에 대한 개선 방안으로 리팩토링 기법을 제안하였다. Table 7은 이들 연구에서 제안한 Energy Bad Smells 및 이에 대한 리팩토링 기법의 간단한 설명을 나타낸다.

Table 7의 리팩토링 기법에서 “Parameter by value”의 경우 어떠한 함수에서 인자로 전달받은 값을 어떠한 처리도 하지 않고 바로 다른 함수에 전달하는 경우를 의미한다. 이는 인자 전달을 위한 행위가 추가되기 때문에 특별한 이유가 없다면 해당 함수를 통합하는 것이 전력 소모 절감효과를 보인다. “Switch redundant assign”의 경우 break문이 존재하지 않기 때문에 특정 조건을 만족하는 경우 해당 조건 이후에 기술된 모든 case문이 실행된다. 의도적으로 이러한 코드를 작성할 수도 있으나, case문이 나타날 때마다 load 와 compare 명령 등이 추가적으로 수행되기 때문에 이러한 경우라면 case문을 통합해주는 것이 효과적이다.

“Complex expressions”의 경우 조건문이 필요 이상으로 복잡하게 작성되어 있어 레지스트리의 값 변경 등을 위한 연산이 추가될 수 있다. 때문에 중첩 분기문 등을 이용하는 것보다 오히려 전력소모량이 더 커질 수 있다는 문제를 가지고 있다. 또한 해당 조건을 확인하는데 있어서 가독성 등이 다소 낮아질 수 있으므로, 유지보수성을 저해하는 요소가 될 수 있다.

“Tail Recursion”의 경우 반복문 등을 통해 해결할 수 있는 코드를 재귀적 호출을 통해 작성한 경우이다. 재귀적 호출은 함수 호출을 위해 이에 대한 지역변수 선언 및 분기 등의 다양한 명령어가 재귀횟수만큼 추가되며, 메모리를 낭비할 수 있다. 때문에 이는 반복문으로 변경해주는 것이 전력 소모량 절감에 용이하다.

“Loop structure”의 경우는 과도한 인덱스 변수 및 반복문 내부에서의 전역변수를 사용하는 경우 등을 의미한다. 특히 전역 변수의 경우 Local Variable Stack에 저장되는 것이 아니라, Global Segment Area에 저장되기 때문에 빈번한 전역변수 접근은 전력소모량을 크게 한다는 단점이 있다. 이러한 경우 불필요한 변수를 제거하고 전역변수에 대한 접근을 최소화 하는 것으로 문제를 해결할 수 있다.

Table 7. Representative Refactoring Techniques for Low-Energy Consumption

Ref.	Energy Bad Smells	Description
[9]	Parameter by Value	Passing parameters to the function without use
	Self Assign	Assign the value with own value (e.g., x=x)
	Mutual Exclusion OR	OR operations has always TRUE value
	Switch Redundant Assign	The switch statement without break
	Deal Local Store	Never used local variable
	Dead Local Store Return	Local variable that is not assigned to return
	Repeated Conditionals	Redundant check of condition statement
	Non Short Circuit	Use the operators &&, || instead of &, | operators
	Useless Control	Control statement does not change the path flow
[10]	Complex Expressions	The codes that contain highly complex expressions
	Common Sub-expressions	The multiple codes that contain the same operation
	Tail Recursion	The codes that is called recursively
	Loop Structure	The loop that can be executed excessively.
	Dead Code	The codes that will not be run at any condition

5. 리팩토링 도구의 특성(Feature)

5.1 리팩토링 도구의 필수 특성

5.1.1 이식성

리팩토링 도구의 선택을 위해서는 우선적으로 이식성이 고려되어야 한다. 이는 다양한 플랫폼에서 동작이 가능해야 함을 의미하며, OS 또는 IDE 툴의 지원 가능 여부를 나타낸다. 어떠한 조직에서 리팩토링 도구를 선정하고자 한다면 해당 조직의 개발 환경과 부합하는지 여부가 고려되어야 하며 아무리 다양한 언어와 기법을 지원한다고 해도 이식성이 낮아 한정적인 환경에서만 동작한다면 이를 활용할 수 있는 기회가 줄어들게 된다. 본 연구에서 살펴본 리팩토링 도구들 중에서는 Eclipse가

가장 많은 플랫폼(Windows, Linux, Apple OS X 등)을 지원하고 있다.

5.1.2 지원 가능한 언어의 다양성

이식성 다음으로 고려해야 할 사항은 지원 가능한 언어의 다양성이다. 물론, 조직에서 사용하는 언어가 단 한 가지로 정해져있다면 다양성의 문제보다는 해당 언어의 지원 여부가 중점이 될 수 있지만, 지원하는 언어가 다양할수록 추후 비즈니스 확장 등에 용이하다는 장점을 갖는다. 본 연구에서 조사한 도구들 중에서는 JAVA 및 C/C++을 지원하는 Eclipse와 C/C++ 및 C# 등을 지원하는 Xcode가 가장 다양한 종류의 언어를 지원하는 것으로 확인되었다.

5.1.3 리팩토링 기법의 지원 범위

리팩토링 도구가 지원하는 환경 및 언어가 다양성을 갖추고 있다고 해도, 실제 지원 가능한 리팩토링 기법이 아주 기본적인 것들만 포함되어 있거나 또는 너무 적은 기법을 지원하고 있다면 실효성이 매우 낮아질 수 있다. 따라서, 리팩토링 도구는 다양한 기법을 지원하고 실제로 해당 기법들을 통해 프로그램 코드의 품질이 향상될 수 있어야만 한다.

5.1.4 확장성

소프트웨어가 만족해야 하는 비기능적 품질 속성들은 수명주기, 적용 도메인 등에 따라 달라진다. 특히 4장에서 언급한 저전력 소모의 경우 기존에는 크게 고려되지 않던 품질 속성이기도 하다. 또한 같은 목적을 가지고 있다 하더라도 다양한 리팩토링 기법들이 계속 연구되고 있다. 따라서 리팩토링 도구는 새로운 리팩토링 기법의 확장이 용이해야 한다.

5.2 기존 도구 특성 분석 및 평가

앞서 리팩토링 도구가 갖추어야 할 속성들을 이식성, 지원성, 확장성 측면에서 살펴보았다. 이 중에서 확장성의 경우, 다양한 Plug-in을 통해 기법의 추가가 용이한 IDE 기반의 지원 기능으로 구성된 도구와 해당 IDE에서 Plug-in으로 동작하는 도구들 간에는 확장의 용이성에 큰 차이가 있으므로, 이를 제외한 나머지 세 가지 속성을 기반으로 Fig. 3에서 Fig. 5와 같이 분류를 수행한다.

Fig. 3은 본 논문에서 제시한 대표적인 리팩토링 도구들을 지원하는 운영환경에 따라 분류한 것이다. 해당 그림은 가장 많이 사용되는 OS인 Windows, Linux 및 MAC OS X를 기준으로 작성하였다. 대부분의 도구들이 단 하나의 운영환경에서 Eclipse (JDT와 CDT를 통합하여 나타냄)와 NetBeans의 경우는 분류 기준인 세 가지 OS를 모두 지원하는 것을 확인 할 수 있다. 이는 해당 도구들이 JAVA 기반으로 개발되었으며, 다양한 플랫폼 상에서 동작이 가능한 JAVA의 특성에 따라 발생하는 차이라고 볼 수 있다.

또한 Visual Studio 기반에서 운영되는 Visual Assist X, ReSharper, Refactor! for C++ 및 Just Code의 경우 MS Windows에 종속되었을 뿐만 아니라 대부분의 경우 Visual Studio에서만 동작하기 때문에 운영 환경에 대한 제약사항이 다소 커지게 된다. 따라서 소프트웨어 개발 조직에서 리팩토링 도구를 활용할 때, 해당 조직의 개발 환경이 다양하게 구성되어 있다면 우선적으로 Eclipse와 NetBeans를 후보로 고려하는 것이 합당하며, Visual Studio를 활용하는 프로젝트를 주로 수행한다면 앞서 언급한 네 가지 리팩토링 도구들을 후보로 식별 할 수 있다. 또한 개발 환경이 Apple 위주로 구성되어 있는 경우에는 Xcode가 가장 적합한 후보로 식별 가능하다.

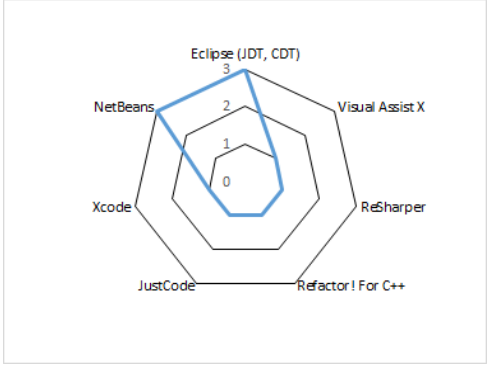


Fig. 3. Supporting Platforms of the Refactoring Tools

Fig. 4는 지원 가능한 언어를 중심으로 비교를 수행한다. 여기서는 Xcode가 가장 많은 언어를 지원하는 양상을 보이는데, 이는 C에서 파생된 다양한 언어들을 지원하기 때문이다. 그 외에 Eclipse의 경우는 JDT와 CDT를 활용하여 (JDT를 이용하더라도 플러그인을 통해 CDT로 확장이 가능) C에서 파생된 언어들은 물론이고 JAVA까지 사용할 수 있다는 장점을 보인다. 이와 마찬가지로 NetBeans 또한 Eclipse와 유사한 언어들을 지원하고 있

다. 반면 Visual Assist X를 포함한 나머지 리팩토링 도구들은 모두 Visual Studio에서 동작함에도 불구하고 지원가능한 언어가 상이하다는 것을 알 수 있다. 때문에 Visual Studio를 통한 개발이 주로 이루어지는 조직에서는 주로 사용하는 언어를 고려하여 해당 리팩토링 도구들로 식별된 후보들을 간추릴 수 있다. 이들 후보들 중에 가장 많은 언어를 지원하는 도구는 Visual Assist X이다.

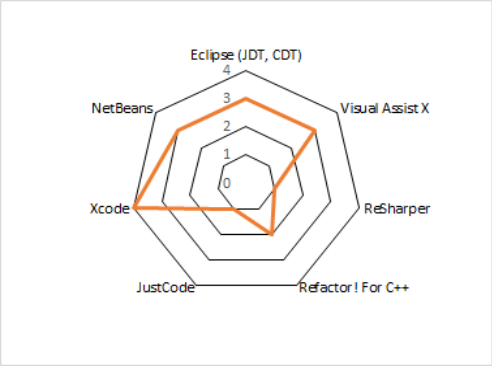


Fig. 4. Supporting Languages of the Refactoring Tools

마지막으로 지원 가능한 리팩토링 기법을 살펴보면 Eclipse가 다른 도구들에 비해 압도적으로 많은 양을 보여주고 있다. Eclipse 재단에서 운영하고 있는 다양한 프로젝트들을 통해 쉽게 해당 기능을 확장할 수 있다는 점에 비추어보면 이러한 차이는 더욱 크게 될 수 있다. 반면 NetBeans의 경우 지원 언어 및 운영 환경 등의 범위가 Eclise와 매우 유사하지만, 지원 가능한 리팩토링 기법의 수에서는 현저하게 적다는 점을 고려할 때, Netbeans보다는 Eclipse를 이용하는 것이 리팩토링 관점에서 다소 유리할 것으로 예상된다.

XCode의 경우 본 논문에서 비교 대상으로 선정한 리팩토링 도구들 중 Objective-C 등을 지원하는 유일한 개발 도구라는 관점에서 Apple OS X에서 개발을 해야 한다면 유일한 선택 사항이 된다. 뿐만 아니라 Eclipse를 제외한다면 지원 가능한 기법의 수도 상당히 많은 편에 속한다. Visual Studio 상에서 동작하는 도구들을 살펴보면 지원 기법의 수가 다소 상이하다. JustCode가 10가지 기법을 지원함으로써 가장 많은 기법을 지원하지만, 앞서 언급한 바와 같이 지원 언어의 범위가 넓지 않다는 점이 단점이다. 때문에 다양한 언어를 지원해야 한다면 Visual Assist X가 오히려 더 용이할 수 있다. 리팩토링 기법에 대한 지원 정도는 Fig. 5에 분석되었다.

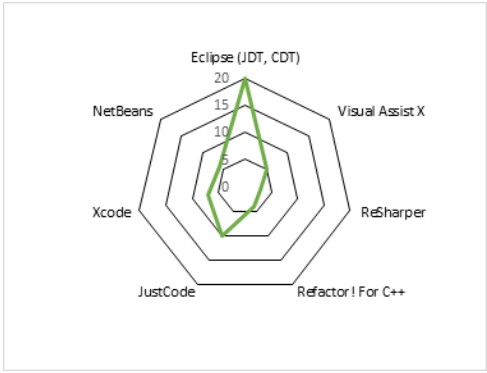


Fig. 5. Supporting Refactoring Techniques of the Refactoring Tools

6. 리팩토링 도구 선정 가이드라인

본 연구에서는 5.1과 5.2를 소프트웨어 개발 프로젝트에 대한 리팩토링 도구의 선정에 있어서 주요 고려사항들에 대하여 식별을 하였다. 앞서 언급한 바와 같이 리팩토링 도구는 그것이 개발도구(IDE)와 통합되어있는 환경을 제공하는 경우도 있으므로, 어떠한 리팩토링 도구를 이용할 것인지에 대한 선택에 앞서 지원 가능한 언어와 리팩토링 기법 및 동작 플랫폼 등에 대한 고려가 선행되어야만 한다. 본 장에서는 해당 요소들 등을 고려하여 리팩토링 도구 선택을 수행하기 위한 가이드라인을 Fig. 6과 같이 제시한다.

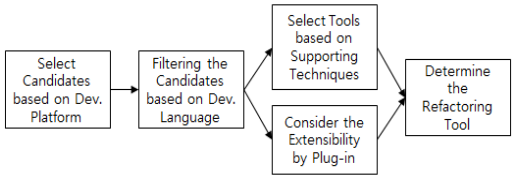


Fig. 6. Guideline to Select A Refactoring Tool

(단계 1) 동작 플랫폼에 따른 후보 선택 : 리팩토링이 필요한 단계는 소프트웨어 개발 단계 중 구현 단계이다. 따라서 이미 운영환경 및 개발환경에 대한 의사결정이 완료되어 있다. 결과적으로 소프트웨어 개발을 수행하고 있는 조직에서는 어떠한 플랫폼에서 동작하는 리팩토링 도구를 선정할 지를 결정하기 위한 정보가 구비되어 있다. 이는 각 리팩토링 도구에서 운영환경 및 개

발환경과 부합하지 않는 도구들을 소거하여 후보군을 선택하는 방법으로 수행 할 수 있다.

(단계 2) 지원 언어에 따른 후보군 정제 : 앞의 단계를 통해 현재의 개발활동에서 사용 가능한 1차 후보군 선정이 완료되었다. 플랫폼에 따른 후보군 선정과 마찬가지로, 구현단계에서는 개발에 사용할 언어 또한 이미 선정이 되어 있는 상태이다. 그러므로 1차 후보군에서 해당 조직이 사용하고자 하는 개발 언어를 지원하지 않는 리팩토링 도구들을 소거하여 1차 후보군을 정제할 수 있다.

(단계 3) 리팩토링 지원 기법에 따른 선정 : 위의 두 단계를 통해 플랫폼과 개발 언어에 부합하는 후보군이 선택되어 있다. 다음으로 고려해야 하는 요소는 후보 군에 속한 각 리팩토링 도구들이 얼마나 다양한 리팩토링 기법들을 지원하는 가에 대한 사항이다. 예를 들어, 앞의 단계를 통해 두 가지 리팩토링 도구가 후보로 남아 있다면 둘 중 더 많은 리팩토링 기법을 지원하는 도구가 현재 프로젝트에 가장 적합한 도구가 될 것이다. 이 과정에서는 확장성을 고려하여, 현재 후보군에 남아있는 도구들 중 특정 도구가 지원 가능한 리팩토링 기법이 다소 적다고 하더라도 plug-in 등을 통해 손쉽게 기능 확장이 가능하다면 해당 도구를 선택 할 수도 있다. 물론 앞의 두 단계를 통해서 선정된 후보군이 단 하나의 리팩토링 도구만을 포함하고 있다면 해당 과정은 생략할 수 있다.

이와 같은 가이드라인을 통해 리팩토링 도구 선정을 한다고 가정할 때, 개발 환경이 Windows이며, JAVA를 통해 개발 한다면 (단계 1)과 (단계 2)를 통해 Eclipse 및 NetBeans를 후보군으로 선정 할 수 있다. 또한 (단계 3)에서는 해당 도구들의 지원 가능한 리팩토링 기법의 다양성을 평가하며, 결과적으로 Eclipse가 주어진 환경에서 보다 효과적으로 리팩토링을 지원할 수 있을 것으로 판단할 수 있다. 이러한 결정에 대해 확장성을 고려하더라도, 리팩토링 기능의 확장이 Eclipse에서 보다 쉽고 다양하게 적용 가능하므로 보다 적합한 리팩토링 도구로 식별이 가능하다. 물론 해당 가이드라인에서 (단계 1)과 (단계 2)는 상황에 따라 서로 순서가 변경 될 수 있다.

7. 결론 및 향후 연구

리팩토링은 기존 레거시 코드의 코드 품질을 향상시키는 중요한 공학적 기법이다. 그럼에도 불구하고 현업의 엔지니어들은 현재 사용중인 레거시 코드에 대한 리팩토링 수행에 주저함을 갖는다. 그 이유는 아마도 리팩토링 기법에 대한 이해 부족과 지원 도구에 대한 지식의 부족에 근거한 것이라 할 수 있다. 따라서 본 논문에서는 기존 리팩토링 기법에 대한 분석과 대표적인 지원 도구들을 조사 분석하여, 이들에 대한 특성을 제시하였다.

리팩토링 도구가 매우 다양한 기법을 지원한다고 하더라도 진행하고자 하는 프로젝트의 환경 등에서 사용이 불가능하다면 이는 실효성이 떨어질 것이며, 또한 다양한 플랫폼에서 동작한다 하더라도 지원되는 리팩토링 기법이 매우 한정적이라면 마찬가지로 실효성이 매우 낮아질 것이다. 따라서 소프트웨어 개발에서 소스 코드의 품질을 향상시키기 위한 리팩토링 도구의 선택은 앞서 언급한 다양한 요소들이 모두 고려되어야만 한다.

향후의 연구는 리팩토링과 관련한 패턴을 개발하는 것이다. 소스 코드가 갖는 구조적 패턴을 기반으로 리팩토링 기법을 자동으로 제안하는 기술을 개발한다면 엔지니어가 리팩토링의 수행에 있어서 주저함을 상당부분 해소시킬 수 있을 것으로 판단한다.

ACKNOWLEDGMENTS

이 논문은 정부(교육부)의 재원으로 한국연구재단-일반연구자지원사업의 지원을 받아 수행된 연구임 (No. NRF-2014R1A1A4A01005566).

REFERENCES

[1] A. Deursen1, P. Klint and C. Verhoef, “Research Issues in the Renovation of Legacy Systems,” *Proceedings of the International Conference on Fundamental Approaches to Software Engineering*, pp. 1-21, 1999.

[2] A. Jatain and D. Gaur, “Reengineering Techniques for Object Oriented Legacy Systems,” *International Journal of Software Engineering and Its Applications*, Vol. 9, No. 1, pp. 35-52, Jan. 2015.

[3] G. Nascimento and C. Iochpe, “A Method for Rewriting Legacy Systems using Business Process Management Technology,” *Proceedings of the the 11th International Conference on Enterprise Information Systems*, pp. 1-6, 2009.

[4] P. Pirkelbauer, D. Dechev and B. Stroustrup, “Source Code Rejuvenation is not Refactoring,” *Proceedings of the 36th Conference on Current Trends in Theory and Practice of Computer Science(SOFSEM 2010)*, LNCS 5901, pp. 639-650, 2010.

[5] E. Murphy-Hill, C. Parnin and A. P. Black, “How We Refactor, and How We Know It,” *Proceedings of the 31st International Conference on Software Engineering(ICSE'09)*, pp. 287-297, 2009

[6] M. Kim, T. Zimmermann and N. Nagappan, “A Field Study of Refactoring Challenges and Benefits,” *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering(FSE'12)*, No. 50, pp. 1-11, 2012.

[7] M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts, *Refactoring: Improving the Design of Existing Code*, Addison Wesley, 2002.

[8] J. D. Garcia and B. Stroustrup, “Improving performance and maintainability through refactoring in C++ 11,” *Proceedings of the ACCU Conference*, pp. 1-20, 2016.

[9] A. Vetro, L. Ardito, G. Procaccianti and M. Morisio, "Definition, Implementation and Validation of Energy Code Smells: an exploratory study on an embedded system," *Proceedings of the Third International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*, pp. 34-39, 2013.

[10] J. Lee, D. Kim, and J. hong, “Code Refactoring Techniques based on Energy Bad Smells for Reducing Energy Consumption,” *Journal of KIPS transactions on software and data engineering*, Vol. 5, No. 5 pp. 209-220, May. 2016.

[11] J. Kim, D. Kim and J. Hong, “Techniques to Support Low-Power Characteristics in Embedded Software Development Process,” *Journal of Convergence Society for SMB*, Vol. 1, No. 1, pp. 55-65, Nov. 2011.

[12] J. Lee and J. Hong “Energy Analysis of System Constraints using SysML Parametric Diagram,” *Journal of Convergence Society for SMB*, Vol. 2, No. 2, pp. 13-19, Dec. 2012.

[13] Eclipse Foundation, “Eclipse CDT,” http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/mars2, 2016. 10.

[14] Telerik, “JustCode,” http://www.telerik.com/products/justcode.aspx, 2016. 10.

[15] Whole tomato software, “Visual Assist,” Available in: http://www.wholetomato.com/, 2016. 10.

[16] Jet brains, “ReSharper C++,” https://www.jetbrains.com/resharper-cpp/, 2016. 10.

[17] W. F. Opdyke and R. E. Johnson, “Refactoring: An Aid in Designing Application Frameworks and Evolving Object-Oriented Systems,” *Proceedings fo the ACM Symposium on Object Oriented Programming Emphasizing Practical Applications*, 1990.

[18] W. G. Griswold, *Program Restructuring as an Aid to Software Maintenance*, Doctoral Dissertation Program restructuring as an aid to software maintenance, 1991. 7.

[19] W. F. Opdyke, *Refactoring Object-Oriented Frame-works*, Doctoral Dissertation Program, University of Illinois at Urbana-Champaign, 1992. 6.

[20] A. Garrido, R. Johnson, “Refactoring C with Conditional Compilation,” *Proceedings of the 18th IEEE International Conference on Automated Software Engineering*, pp. 1-4, 2003.

[21] L. Hunt, *C# Coding Standards for .NET*, Document Version, 2007. 3.

[22] M. Gatrell, S. Counsell and T. Hall, “Empirical support for two refactoring studies using commercial C# software,” *Proceedings of the 13th International Conference on Evaluation & Assessment in Software Engineering*, pp. 1 - 10, 2009.

[23] Apple Developer, “Adopting Modern Objective-C: Guide and Sample Code,” https://developer.apple.com/library/content/releasenotes/ObjectiveC/ModernizationObjC/AdoptingModernObjective-C/AdoptingModernObjective-C.html, 2014. 10.

[24] M. Schafer, J. Dolby, M. Sridharan, E. Torlak and F. Tip, “Correct Refactoring of Concurrent Java Code,” *Proceedings of the 24th European conference on Object-oriented programming(ECOOP '10)*, LNCS-6183, pp. 225-249, 2010

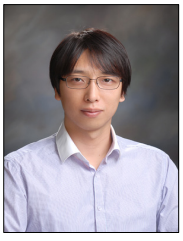
[25] D. Dig, J. Marrero and M. D. Ernst, “Refactoring Sequential Java Code for Concurrency via Concurrent Libraries,” *Proceedings of the 31st International Conference on Software Engineering*, pp.397-407, 2009.

[26] C. W. Fraser, E. Mayers and A. L. Wendi, “Analyzing and Compressing Assembly Code,” *Proceedings of the*

ACM SIGPLAN '84 Symposium on Compiler Construction(SIGPLAN '84), Vol. 19, No. 8, pp. 117-121, 1984.
[27] A. Fog, *Optimizing subroutines in assembly language*, Technical University of Denmark, 2015.
[28] Eclipse Foundation, "Eclipse JDT," Available in: https://eclipse.org/downloads/packages/eclipse-ide-java-developers/mars2, 2016. 10.
[29] Semantic designs, "Refactor! for C++," http://www.semanticdesigns.com/Products/DMS/Refactoring.html, 2016. 10.
[30] Apple, "Xcode," https://developer.apple.com/xcode/, 2016. 10.
[31] Oracle, "NetBeans," https://netbeans.org/, 2016. 10.
[32] M. Vakilian, "Compositional Refactoring," https://marketplace.eclipse.org/content/compositional-refactoring, 2016. 10.
[33] J. N. Rouvignac, "AutoRefactor," https://marketplace.eclipse.org/content/autorefactor, 2016. 10.
[34] J. Reimann, "Refactory," https://marketplace.eclipse.org/content/refactory, 2016. 10.
[35] Y. Gil, "Spartan Refactoring," https://marketplace.eclipse.org/content/spartan-refactoring-0, 2016. 10.

저 자 소 개

김 두 환(Kim, Doohwan) [정회원]



- 2007년 2월: 충북대학교 컴퓨터공학과 학사
- 2009년 2월 : 충북대학교 컴퓨터과학과 석사
- 2016년 2월 : 충북대학교 컴퓨터과학과 박사
- 2016년 3월 ~ 현재 : 충북대학교 소프트웨어공학연구실 포닥 연구원 및 초빙 교원

<관심분야> : 소프트웨어 품질, 소프트웨어재사용, 저전력 소프트웨어 개발, 코드리팩토링

정 유 진(Jeong, Yu-Jin) [정회원]

- 2013년 3월: 충북대학교 소프트웨어학과 학사 입학
- 2016년 현재 : 충북대학교 소프트웨어학과 학사 재학중

<관심분야> : 웹앱 기반 소프트웨어, 소프트웨어 리팩토링, 데이터 기반 모델링 기법

홍 장 의(Hong, Jang-Eui) [종신회원]

- 2001년 2월 : KAIST 전산학과 공학박사
- 2002년 10월 : 국방과학연구소 정보체계연구부 선임연구원
- 2004년 8월 : ㈜솔루션링크 기술연구소 소장
- 2004년 9월 ~ 현재 : 충북대학교 소프트웨어학과 교수

<관심분야> : 융합 소프트웨어, 모델기반 소프트웨어공학, 소프트웨어 품질, 저전력 소프트웨어 개발