The Journal of Digital Policy & Management 2013 Jul; 11(7): 165-171

소스 코드에 포함된 코딩 패턴의 특성과 출현 위치 관련성에 대한 분석

김영태*, 공헌택*, 김치수*
공주대학교 컴퓨터공학과*

Analysis of characteristics and location of the appearance for codding pattern in the source code

Young-Tae Kim*, Heon-Tag Kong*, Chi-Su Kim*
Dept. of Computer Science & Engineering*

요 약 코딩 패턴이란 소스 코드에 자주 나타나는 전형적인 코드 조각이다. 소프트웨어에서 로깅이나 동기화 처리 등 모듈화가 곤란한 기능과 프로그래밍의 상용구 등이 코딩 패턴으로 추출된다. 기존에 소스 코드에 대한 패턴 마이닝을 이용한 코딩 패턴 감지 방법이 제안되었지만, 수동으로 조사 가능한 코딩 패턴의 수는 한정되어 있기 때문에 대규모 소프트웨어 등에 대한 충분한 분석은 할 수 없다.
본 논문에서는 개발자가 분석하고자 하는 코딩 패턴만을 추출하는 것을 목표로 코딩 패턴의 특성 평가 척도로 6개의 지표를 선정하여 4개의 오픈 소스에 대한 분석을 실시했다. 매트릭스 사이의 값의 관계와 실제 패턴의 특징을 분석한 결과, 패턴의 인스턴스 수, 인스턴스의 분포의 넓이, 패턴의 요소에 포함되는 반복 구조 비율 등의 지표 등을 분석해야 패턴 선택에 도움이 되는 것을 확인했다.

주제어 : 코딩 패턴, 소프트웨어 매트릭스

Abstract Coding patterns that appeared frequently in the source code is a typical piece of code. The functionality that difficult to modularize, such as logging or synchronization processing, and the useful sentences in programming is extracted in software as codding pattern. Large-scale software could not be analyzed fully because the number of coding pattern that can be manually investigated is limited.
In this paper, the characteristics of coding patterns perform the evaluation. The goal is to extract for codding-pattern to analyzed by developer. We was selected 6 indicators and performed analysis of 4 open-source. Matrix relations between the values and characteristics of the actual pattern analysis, pattern instances, the width of the distribution of instances, the pattern repeating structure of the elements included in the rates should be analyzed for patterns and indicators that help in choosing was confirmed.

Key Words : Coding Pattern, Software Metrics

* 본 논문은 공주대학교 연구년 사업에 의하여 지원되었음
Received 21 May 2013, Revised 20 June 2013
Accepted 20 July 2013
Corresponding Author: Chi-Su Kim (Div. of Computer Science & Engineering)
Email: cskim@kongju.ac.kr

ISSN: 1738-1916

1. 서론

객체 지향 프로그래밍에서는 상속과 다형성의 구조를 이용하여 소프트웨어의 재사용성과 유지 보수성을 향상시킬 수 있다. 그러나 객체 지향 구조에서는, 모듈화하기 어려운 기능이 존재하며, 이러한 기능의 구현은 소스 코드에 반복적으로 등장한다[1]. 대표적인 예로, 로깅 및 동기화를 들 수 있으며, 기능에 해당하는 소스 코드가 여러 모듈에 횡단적으로 나타나기 때문에 횡단적 관심사라고도 불린다[2].

여러 모듈에 분산 배치되는 코드는 원본이 되는 소스 코드 조각을 개발자가 복제, 배포하여 상황에 따라 적절하게 수정을 추가하는 방식으로 만들어지는 경우가 많으며, 무리를 이루는 전형적인 코드 조각, 즉 코딩 패턴을 구성한다. 코딩 패턴에 속하는 코드 조각은 서로 유사하며, 또한 많은 동일한 기능을 제공하고 있다. 따라서 코드 조각 한 개를 변경하는 경우 동일한 패턴에 속하는 다른 코드 조각에 대해서도 일관된 규칙을 적용할 것인지 검토할 필요가 있다[3, 4].

소스 코드에 대한 패턴 마이닝을 이용한 코딩 패턴 감지 방법을 제안한 연구들이 있다[5]. 그러나 대규모 소프트웨어에서는 수많은 코딩 패턴이 발견되고, 분석을 수동으로 진행하기 때문에, 조사 가능한 패턴의 수가 매우 한정된다. 호출 횟수가 많은 메서드는 횡단적인 관심사와 관련이 있다는 Marin 등의 지적[6]에 따라, “패턴에 해당하는 원시 코드 조각의 수가 많을수록 중요한 패턴”이라는 일원적인 평가 척도에 의해, 분석 대상의 패턴을 선택해왔다. 패턴에 해당하는 소스 코드 조각의 수는, 종종 유용한 패턴 발견에 도움이 되지만, 언어 사양 혹은 코딩 스타일 등에서 우연히 패턴에 해당하는 코드 조각이 발생하기도 하므로, 의미가 없는 패턴을 수동으로 제거할 필요가 있다.

본 논문에서는 개발자가 주목하고 싶은 패턴만 효율적으로 분석 가능한 환경을 구축하기 위해 새로운 평가 척도로 도입 가능한 매트릭스 평가를 실시한다. 패턴의 길이와 인스턴스 변경 수, 한 패턴에 포함된 요소 종류의 수 등의 간단한 지표 이외에 코드 클론 검출 방법[7]에서 사용한 소스 코드 조각의 출현 위치에 대한 매트릭스[6]에 관하여도 평가를 실시한다.

2장에서는 코딩 패턴과 코딩 패턴 검색 기술에 대해 설명하고, 3장에서는 코딩 패턴의 특징, 외관 위치에 대해 설명한다. 그리고 4장에서는 행한 실험과 그 결과에 관하여 언급하고, 마지막으로 5장에서는 결론 및 향후연구 과제를 언급한다.

2. 코딩 패턴

코딩 패턴이란 여러 개의 모듈에 분산되어 있는 전형적인 코드이다. 본 논문에서는 코딩 패턴을 메서드 호출 요소와 그에 따른 제어 구조 요소(조건 분기와 반복)의 전형적인 열이라고 파악하는 패턴 마이닝 기법을 기반으로 한다[5].

2.1 코딩 패턴 예

Java에서는 컬렉션 개체에 포함되는 각 요소를 처리하기 위해서, 디자인 패턴의 일종인 Iterator 패턴[8]을 이용할 수 있다. 반복처리는 실제의 소스 코드에서 for 문이나 while 문으로 기술되지만, 반복 처리의 정규화 프로세스를 통해 “LOOP”와 “END-LOOP”라는 쌍으로 표현된다. [Fig. 1]은 그림 편집기 JHotDraw 5.4b1에서 추출된 Undo 패턴이다. Undo 패턴의 인스턴스 부분에서 실제로 수행하는 프로세스는 다르지만, 밑줄로 표시되는 메서드 호출 열이 같다. 이러한 메서드 호출 패턴을 알고 있는 것이, “Undo 처리”의 구현을 이해하는데 도움이 된다. 또한 새로운 편집 작업을 구현하는 데 유용하다.

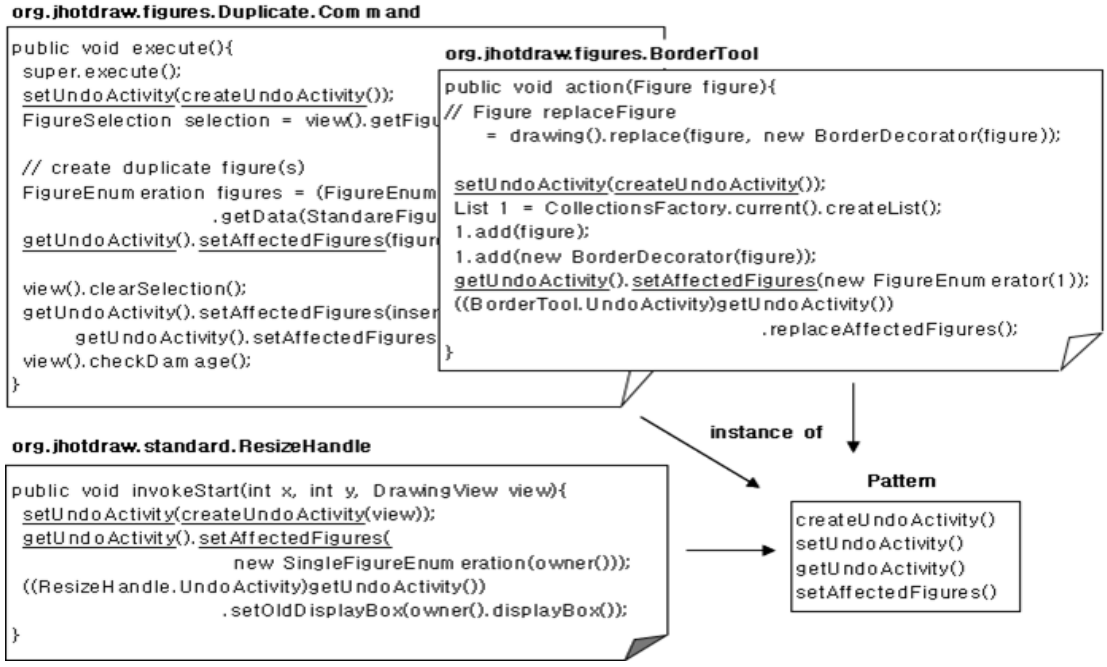
2.2 코딩 패턴 추출법

코딩 패턴의 추출은 먼저 분석 대상의 원시 코드를 특징 순서를 추출하는 단위인 메서드로 분할한다. 다음에 분할된 각 메서드 중에서 특징 정보를 추출하여 특징 데이터베이스에 등록한다. 그리고 특징 데이터베이스에 대해서, 패턴 마이닝을 적용한 코딩 패턴을 추출한다. [Fig. 2]는 코딩 패턴 추출 흐름을 보여 준다.

본 논문에서는 Java로 작성된 소스 코드에서 코딩 패턴의 추출을 실시한다.

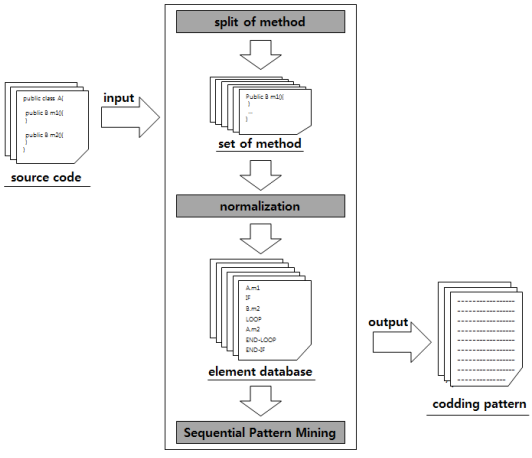
2.2.1 소스 코드의 정규화

소스 코드의 정규화는 Java 프로그램의 메서드를 별도의 코드 조각으로 간주하고, 메서드 단위로 메서드 호



[Fig. 1] Undo pattern that has been extracted from the JHotDraw 5.4b1

출 요소와 제어 구조 요소의 문자열로 변환한다. Java 프로그램은 일반적으로 다수의 메서드로 구성되어 있기 때문에 정규화에 의해서 Java 프로그램 요소 열의 데이터베이스로 변환된다. 이 요소 열 데이터베이스가 다음 단계인 패턴 마이닝의 대상이 된다.



[Fig. 2] Flow of coding pattern extraction

메서드의 정규화 작업은 다음 3 가지로 구성된다.

- 메서드 호출의 정규화: 소스 코드에 등장하는 메서드 호출 식을 메서드 호출 요소로 변환한다. 메서드 호출 요소는 메서드 이름, 반환형식, 인수형식 이름 등의 문자열을 유지한다.
- 조건부 정규화: 조건 분기의 정규화를 실시하는 것으로 소스 코드의 if 문과 삼항 연산자는 조건 분기로 정규화 한다.
- 반복 처리의 정규화: 메서드에 등장하는 for 문, while 문, do-while 문장을 정규화 한다.

2.2.2 순차 패턴 마이닝

순차 패턴 마이닝은 대상 요소 열에서 요소의 순서를 고려하여 패턴을 검출하는 방법이다. 본 논문에서는 소스 코드의 정규화를 실시해 작성한 특징 데이터베이스에 대해서 순차 패턴 마이닝 알고리즘의 하나인 PrefixSpan[9]을 적용해 코딩 패턴을 추출한다.

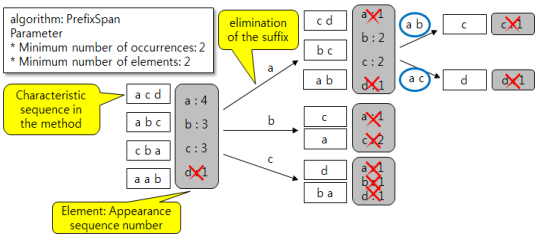
[Fig. 3]은 순차 패턴 마이닝 처리의 예이다. 4 개의 특징 시퀀스에서 요소 수 2 이상의 패턴을 검출한다. 순차 패턴 마이닝에서 먼저 시퀀스에 등장하는 모든 요소에

대해 요소가 나타나는 수를 조사한다. [Fig. 3]에서 요소 a는 4개의 시퀀스에 등장한다. 여기에서 요소 수 2 이상 조건에 따라 필터링을 실시한다.

2.2.3 불필요한 패턴 삭제

패턴 마이닝의 결과는 패턴으로서 부적절한 것이 포함되어 있다. 다음과 같은 겨우 부적절하다고 판단하여 패턴 마이닝 결과에서 제거한다.

- 제어 구조의 시작 요소와 종료 요소 간에 메서드 호출이 포함되지 않는 패턴
- 제어 구조의 비율이 70% 이상 패턴
- 제어 구조의 시작 요소와 종료 요소가 대응 되지 않는 패턴
- ELSE 요소가 IF 요소와 END-IF 요소로 둘러싸여 있지 않은 패턴



[Fig. 3] Sequential pattern mining

3. 코딩 패턴의 특징과 출현 위치

코딩 패턴은, 소스 코드에 자주 나타나는 메서드 호출의 열이다. 한 개의 프로그램에서 수백 수천의 코딩 패턴을 추출해야 하므로 기존에는 분석 대상이 인스턴스 수가 많은 패턴 순으로 선택한, 극히 소수의 패턴에 한정되었다[5]. 인스턴스 수가 많은 패턴을 우선적으로 조사한 것은 횡단적 관심사는 호출되는 횟수가 많은 메서드에 의해서 구성되어 있는 것이 많다는 Marin 등의 지적에 따른 것이다[6].

코딩 패턴을 효과적으로 분석하기 위하여 분석가가 주목해야 할 코딩 패턴과 인스턴스만을 자동으로 추출하는 것이 중요하다. 본 논문을 그 기초로 사용할 수 있다. 소프트웨어 매트릭스 후보를 선정하여 코딩 패턴에 적용하여 특성을 조사했다.

패턴 분석을 목적으로 모든 패턴 P를 취합하여 정수 또는 실제 숫자를 반환하는 함수 f(P) : Pattern → value를 정의한다. 패턴 P에 대해 이러한 인스턴스 i는 $i \in P$와 같이 집합 P의 원소로 나타내고, 패턴 P의 인스턴스 수는 |P|로 기술한다.

3.1 패턴 길이 : LEN (Pattern Length)

코딩 패턴 P의 패턴 길이 LEN(P)는 패턴 P에 포함된 요소의 수를 나타내는 정수이다. 코딩 패턴 탐지 도구를 사용하여 패턴 길이의 임계값 미만 패턴은 검색 결과에서 제거한다.

3.2 패턴의 인스턴스 수 : NOI (Number of Instances)

코딩 패턴 P의 인스턴스 수를 $\mathrm{NOI}(\mathrm{P}) = |\mathrm{P}|$는 패턴 P를 구성하는 요소 열이 마이닝되는 소스 코드에 나오는 횟수를 나타내는 정수이다. 본 논문에서 사용하는 패턴 마이닝 알고리즘 Pre xSpan에서는 임계값으로 이용된다.

3.3 제어 구조 요소의 비율 : RCE (Ratio of Control Elements)

코딩 패턴 P의 제어 구조 요소의 비율 RCE(P)는 패턴 P에 포함된 모든 요소에 따른 제어 요소 수를 비율로 계산하는 실수 값이다.

RCE(P)의 값이 큰 패턴 P는 메서드 호출을 포함하지 않는 if 문 또는 for 문을 간단한 중첩 관계만을 표현했을 가능성이 높다. 따라서 패턴 마이닝 후, 경험적으로 정한 임계값 $\mathrm{RCE}(\mathrm{P}) \leq 0.7$ 조건으로 패턴 필터링을 실시한다.

3.4 패턴의 밀도 : DEN (Density)

코딩 패턴 P의 밀도 DEN (P)는 패턴 P의 각 요소가 소스 코드에 얼마나 조밀하게 배열되어있는지를 나타내는 실수 값이다. 다음의 식에 의해 계산한다.

$$DEN(P) = \frac{\Sigma_{i \in p} DEN_{inst}(i)}{|P|}$$

DENinst(i)는 패턴 인스턴스의 밀도이다. 패턴에 해당

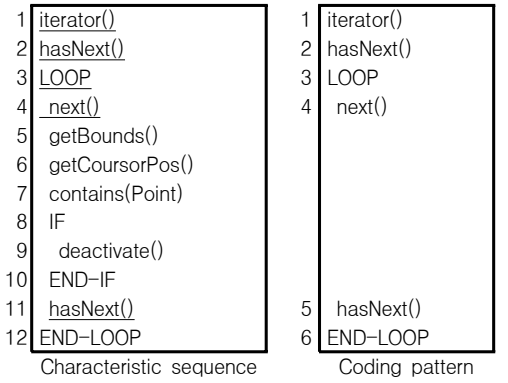
하는 요소를 포함하는 메서드의 요소 열이 주어졌을 때

$$DEN_{inst}(i) = \frac{LEN(P)}{i\text{의 마지막 요소 위치} - i\text{의 시작 요소 위치} + 1}$$

이다.

[Fig. 4]의 오른쪽 부분의 코딩 패턴이 왼쪽과 같은 인스턴스로서 출현할 때, 선두 요소 iterator()에서 마지막 요소 END-LOOP까지 패턴에 해당하는 6개의 요소가 출현한다고 간주하면, 이 인스턴스에 대한 밀도 $DEN_{inst}(i) = 6/(12-1+1) = 0.5$가 된다.

패턴 P의 밀도 DEN(P)는 P의 모든 인스턴스의 밀도의 평균으로 정의되므로, 각 인스턴스가 다른 요소를 포함하지 않는 단일 코드 조각에 가까울수록 값이 1에 가까워진다.



[Fig. 4] Metrics DEN

3.5 반복되지 않는 요소 비율 : RNR(Ratio of Non – Repeated Elements)

패턴 P의 "비 반복"을 의미하는 RNR(P)는 패턴 P의 요소에 포함되는 반복 구조를 검출, 제거 후 남아있는 요소의 비율을 보인 실수 값이다. 본 논문에서 반복 검색은 SEQUITUR 알고리즘[10]을 이용한다. 이 알고리즘은 어느 요소 열이 주어졌을 때, 2개 요소의 집합이 2회 이상 출현하는 경우를 반복으로 검출한다.

3.6 패턴 인스턴스의 분산 : RAD (Radius)

패턴 인스턴스의 분산 RAD(P)는 패턴 P의 인스턴스가 배포되는 소스 코드의 범위를 나타내는 정수이다.

4. 매트릭스를 이용한 코딩 패턴 분석

코딩 패턴의 특성을 조사하기 위하여 6가지 매트릭스를 오픈 소스에서 추출한 코딩 패턴에 적용했다. <Table 1>은 적용 대상 소프트웨어에 대한 정보이다.

〈Table 1〉 Target software

software name	version	Scale(LOC)	Total pattern
JHotDraw	7.0.9	90166	375
jEdit	4.3pre10	168335	2902
Apache Tomcat	6.0.14	313479	8782
SableCC	3.2	35388	450

4.1 매트릭스 사이의 관련성 분석

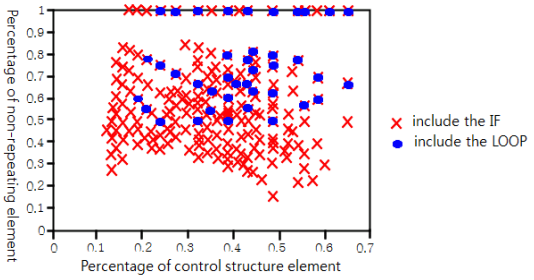
본 논문에서는 6가지 매트릭스의 모든 조합에 대해 관련 조사를 실시했다. 여기서는 관련성이 인정되는 매트릭스의 조합만 언급한다. <Table 2>에 제어 구조 요소를 포함하는 패턴과 제어 구조 요소가 없는 패턴의 수를 보인다.

〈Table 2〉 Number of patterns that do not contain pattern including a control structure

software name	Without control structural element	There control structure elements
JHotDraw	140	235
jEdit	1212	1690
Apache Tomcat	2489	6293
SableCC	120	330

4개의 소프트웨어 모두에서 제어 구조 요소를 포함한 패턴의 수가 제어 구조를 포함하지 않는 패턴의 수보다 많이 검출되었다. 제어 구조 요소가 없는 패턴은 반복되지 않는 요소의 비율에 편향이 없고 넓게 분포하고 있기 때문에, 제어 구조 요소를 포함하는 패턴에만 주목한다.

[Fig. 5]에서 조건부 요소를 포함하는 요소는 전체로 확산되고 있지만, 반복 요소는 반복되지 않는 요소의 비율이 1에 가까운 쪽으로 치우쳐 분포하고 있다.



[Fig. 5] Percentage of non-repeating elements and percentage of control structure element (Apache Tomcat)

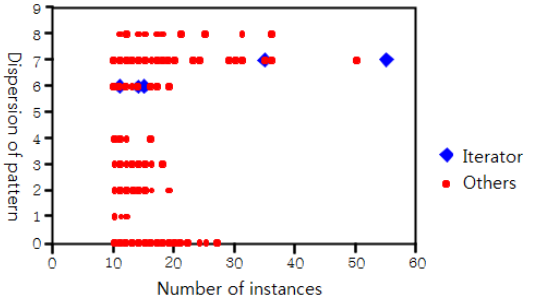
종합적으로 판단해보면, 제어 구조 요소, 특히 그 중에서도 LOOP 구조의 요소를 포함하는 패턴은 반복되지 않는 요소의 비율이 높아지는 경향이 있다. 패턴 내에 LOOP 구조를 갖는다는 것은, 반복 처리가 그 LOOP 구조에 의해 집약 된 것을 의미하므로, 반복되는 요소가 감소하고 반복되지 않는 요소의 비율이 높아진다고 생각할 수 있다.

4.2 반복 출현 위치 관련

코딩 패턴에서 다음 조건을 만족하는 코딩 패턴을 반복을 사용하는 패턴으로 추출한다.

- "next"를 포함하는 메서드 호출 요소가 있다.
- "hasNext"를 포함하는 메서드 호출 요소가 있다.
- "LOOP", "END - LOOP" 반복 작업을 포함한다.

인스턴스 수와 패턴 분산간의 관계에 대한 예로 JEdit의 경우를 [Fig. 6]에 보였다.



[Fig. 6] Relevance of the number of instances and variance of pattern(jEdit)

전통적인 코딩 패턴 조사 방법은 인스턴스가 많은 것부터 조사했다. 그러나 JHotDraw와 jEdit처럼 반복을 사용하는 패턴이 인스턴스 수가 많은 부분에 등장하기 때문에 유익한 패턴이 많은 코딩 패턴 사이에 묻혀 버리는 것도 생각할 수 있다. 이 문제를 해결하기 위해 분산이 낮은 패턴, 즉, 특정한 패키지나 파일에만 등장하는 패턴 순으로 조사하는 등의 방법으로 전환할 필요가 있다.

또한 소프트웨어의 전체 패키지 계층이 없는 소프트웨어에 관해서는 판단이 어렵기 때문에 패턴의 출현 위치 정보로 이용하는 매트릭스를 개량할 필요가 있다.

5. 결론 및 향후연구

코딩 패턴을 유형별로 분류하고, 분석하려는 개발자가 필요로 하는 코딩 패턴만을 선별 제시할 필요가 있다. 따라서 본 논문에서는 코딩 패턴의 특성을 측정하기 위한 지표를 제시하고 그 특징과 관련한 코딩 패턴의 종류와 관계에 대해 분석했다. 그 결과 패턴의 인스턴스 수, 인스턴스의 분포, 패턴의 요소에 포함되는 반복 구조의 비율 등의 지표들이 분석해야 할 패턴의 선택에 있어서 매우 유용하다는 것을 확인했다.

향후연구로는 분석 결과를 바탕으로 코딩 패턴 필터링 기술의 실현을 목표로 한다. 또한 코딩 패턴을 효과적으로 개발자에게 제시하기 위해 코딩 패턴 열람 환경을 통합 개발 환경에서 제공하거나, 코딩 패턴에서 소스 코드의 이용 방법을 추출하여 소프트웨어 컴포넌트로 제공하는 것 등을 들 수 있다.

ACKNOWLEDGMENTS

This study was supported by the research year program of Kongju National University.

REFERENCES

[1] M. Marin. Reasoning about assessing and improving the seed quality of a generative aspect

mining technique. In Proceedings of the 2nd International Linking Aspect Technology and Evolution Workshop, 2006.

[2] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. Loingtier, andJ. Irwin. Aspect oriented programming. In Proceedings of the 11th European Con-ference on Object-Oriented Programming, 1997.

[3] Martin Fowler. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999.

[4] B. S. Baker. A program for identifying duplicated code. Computing Science and Statistics, Vol. 6, 1992.

[5] Takashi Ishio, Hironori Date, Tatsuya Miyake, Katsuro Inoue. Mining coding patterns to detect crosscutting concerns in java programs. In Proceedings of the 15th IEEE Working Conference on Reverse Engineering, 2008.

[6] M. Marin, A. van Deursen, L. Moonen. Identifying aspects using fan - in analysis. In Proceedings of the 11th Working Conference on Reverse Engineering, 2004.

[7] T. Kamiya, S. Kusumoto, K. Inoue. CCFinder: A multi-linguistic token-based code clone detection system for large scale source code. IEEE Transactions on Soft-ware Engineering, Vol. 28, No. 7, 2002.

[8] Erich Gamma, 김정아 역, GOF의 디자인 패턴, 2007.

[9] J. Pei, J. Han, B. Mortazavi - Asl, H. Pinto, Q. Chen, U. Dayal, M. Hsu. Pre - xspan : Mining sequential patterns by pre x - projected growth. In Proceedings of the 17th International Conference on Data Engineering, 2001.

[10] CG Nevill - Manning, IH Witten. Identifying hierarchical structure in sequences : A linear - time algorith. Journal of Arti cial Intelligence Research, 1997.

김 영 태(Kim, Young Tae)

- 2000년 2월 : 공주대학교 전자계산학과 졸업(학사)
- 2002년 2월 : 공주대학교 전자계산학과 졸업(석사)
- 2004년 2월 : 공주대학교 컴퓨터공학과 박사과정 수료
- 관심분야 : 데이터통합, UML, 온톨로지
- E-Mail : zerot@kongju.ac.kr

공 헌 택(Kong, Heon Tag)

- 1984년 5월 : Northeast Missouri State Univ. 전산학과(학사)
- 1987년 12월 : Utah State Univ. 전산학과(석사)
- 1998년 2월 : 단국대학교 전산통계학과(박사)
- 1988년 1월 ~ 1990년 3월 : 한국국방연구원 전산체계연구부 근무
- 1990년 4월~현재 : 공주대학교 컴퓨터공학부 교수
- 관심분야 : 병렬 알고리즘, 병렬처리 컴퓨터, 데이터베이스
- E-Mail : htkong@kongju.ac.kr

김 치 수(Kim, Chi Su)

- 1984년 2월 : 중앙대학교 전자계산학과(학사)
- 1986년 8월 : 중앙대학교 전자계산학과(석사)
- 1990년 8월 : 중앙대학교 전자계산학과(박사)
- 1990년 9월 ~ 1992년 8월 : 공주교육대학교 전임강사
- 1992년 9월 ~ 현재 : 공주대학교 컴퓨터공학부 교수
- 관심분야 : 소프트웨어공학, 소프트웨어개발방법론
- E-Mail : cskim@kongju.ac.kr