소스 코드 분석을 통한 코딩 패턴의 특성 평가에 관한 연구

김영태*, 임재현*, 공헌택*, 김치수*
*공주대학교 컴퓨터공학부
zerot@kongju.ac.kr

A Study on the Evaluation of Coding Pattern Quality Using Source Code Analysis

Young-Tae Kim*, Jae-Hyun Lim*, Heon-Tag Kong*, Chi-Su Kim*
*Dept of Computer Engineering, KongJu National University

요 약

본 논문에서는 소스 코드에 자주 나타나는 코딩 패턴을 자동으로 추출하기 위하여 소스 코드를 분석하고 특성을 평가한다. 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 없었다. 따라서 본 논문에서는 개발자가 분석하고자 하는 코딩 패턴에 대한 자동 추출을 목표로 코딩 패턴의 특성 평가 지표를 선정하여 소스 코드에 대한 분석을 수행한다.

1. 서론

최근 소프트웨어 개발에 객체 지향 프로그래밍의 사용이 증가하고 있다. 객체 지향의 특징인 상속과 다형성의 구조를 이용하여 소프트웨어의 재사용성과 유지 보수성을 향상시킬 수 있다. 그러나 객체 지향의 구조에서는, 모듈화하기 어려운 기능이 존재하며, 이러한 기능의 구현은 소스 코드에 반복적으로 등장한다[13]. 이러한 기능의 대표적인 예로, 로깅 및 동기화를 들 수 있으며, 기능에 해당하는 소스 코드가 여러 모듈에 횡단적으로 나타나므로 횡단적 관심사라고도 불린다[1].

이러한 여러 모듈에 분산 배치되는 코드는 원본이 되는 소스 코드 조각을 개발자가 복제, 배포하여 앞으로 상황에 따라 적절하게 수정을 추가하는 방식으로 만들어지는 경우가 많으며, 무리를 이루는 정형적인 코드 조각, 즉 코딩 패턴을 구성한다. 코딩 패턴에 속하는 코드 조각은 서로 유사하며, 또한 많은 동일한 기능을 제공하고 있다. 때문에 코드 조각 1개를 변경하는 경우 개발자는 동일한 패턴에 속하는 다른 코드 조각에 대해서도 일관된 규칙을 적용할 것인지 검토할 필요가 있다[2].

호출되는 횟수가 많은 메서드는 횡단적인 관심사와 관련이 있다는 지적[3]에 따라, 패턴에 해당하는 원시 코드 조각의 수가 큰 것만큼 중요한 패턴이다, 라고 하는 일원적인 평가 척도에 의해, 분석 대상의 패턴을 선택하고 있었다. 패턴에 해당하는 소스 코드 조각의 수는, 종종 유용한 패턴 발견에 도움이 되지만, 언어 사양 혹은 코딩 스타일등에서, 우연히 패턴에 해당하는 코드 조각이 발생하기도 해, 의미가 없는 패턴을 수동으로 제거할 필요가 있었다.

본 논문에서는 개발자가 주목하고 싶은 패턴만 효율적으로 분석 가능한 환경을 구축하기 위해 새로운 평가 척도로 도입 가능한 매트릭스 평가를 실시했다. 패턴의 길이와 인스턴스 변경 수, 1개 패턴에 포함된 요소 종류의 수 등의 간단한 지표 이외에 코드 복제 검출 방법[10]에서 사용한 소스 코드 조각의 출현 위치에 대한 매트릭스[6]에 관하여도 평가를 실시했다.

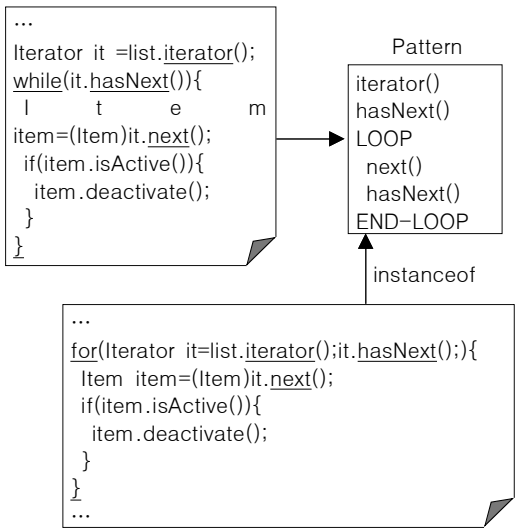
2. 관련 연구

2.1 코딩 패턴

코딩 패턴이란 여러 개의 모듈에 분산되어 있는

정형적인 코드이다. 본 논문에서는 코딩 패턴을 메서드 호출 요소와 그에 따른 제어 구조 요소의 전형적인 열이라고 파악하는 패턴 마이닝 기법을 기반으로 한다[4].

Java에서는 컬렉션 개체에 포함되는 각 요소를 처리하기 위해서, 디자인 패턴[5]의 일종인 Iterator 패턴을 이용할 수 있다. 일련의 과정이 [그림 1]과 같이 Iterator를 사용하여 반복 패턴으로 추출된다. 반복처리는, 실제의 소스 코드에서는, for 문이나 while 문으로 기술되지만, 반복 처리의 정규화 프로세스를 통해 "LOOP"와 "END - LOOP"라는 쌍으로 표현된다.



[그림 1] Iterator를 사용한 반복 패턴

Undo 패턴은, JHotDraw 5.4b1 사용자가 작업을 취소하는 "Undo 처리"를 실현하는 패턴이다.

이러한 메서드 호출 패턴을 개발자가 알고 있는 것이, 이 "Undo 처리"가 어떻게 구현되어 있는지 이해하기 위해 도움이 된다. 또한 새로운 편집 작업을 구현하는 데 유용하다.

2.2 코딩 패턴 추출법

소스 코드에 대한 패턴 마이닝을 이용한 코딩 패턴 감지 방법을 제안한 연구에서 코딩 패턴은 프로그램의 횡단적 관심사에 해당하는 패턴뿐만 아니라 라이브러리의 정형적인 사용법 등도 포함하고 있는 것으로 판명되었다[4].

그러나 대규모 소프트웨어는 수많은 코딩 패턴이 발견되지만, 그 분석을 수동으로 의존하고 있기 때문에, 조사 가능한 패턴의 수가 매우 한정되어 있다.

본 논문에서는 개발자가 주목하고 싶은 패턴만 효율적으로 분석 가능한 환경을 구축하기 위해 새로운 평가 척도로 도입 가능한 매트릭스 평가를 실시했다. 패턴의 길이와 인스턴스 변경 수, 하나의 패턴에 포함된 요소 종류의 수 등의 간단한 지표 이외에 코드 클론 검출 방법[10]에서 사용한 소스 코드 조각의 출현 위치에 대한 메트릭스[6]에 관하여도 평가를 실시했다.

3. 코딩 패턴의 특징과 출현 위치

기존 연구에서는, 개발자가 부품화하는 것이 곤란한 "횡단적 관심사"에 해당하는 기능을 발견하기 위하여 코딩 패턴을 분석했으나, 하나의 프로그램에서 많은 코딩 패턴을 추출해야 하기에 그 만큼 분석 대상은 인스턴스수가 많은 패턴으로부터 차례로 선택한, 극히 소수의 패턴에 한정되어 있었다[4]. 인스턴스수가 많은 패턴을 우선적으로 조사한 것은, 패턴 중에서 호출되는 메소드는 호출되는 횟수가 많다고 하는 사실과, 횡단적 관심사는 호출되는 횟수가 많은 메소드에 의해서 구성되어 있는 것이 많다는 Marin 등의 지적에 따른 것이다[3].

코딩 패턴을 효과적으로 분석하기 위하여 분석가가 주목해야 할 코딩 패턴과 인스턴스만을 자동으로 추출하는 것이 중요하다. 본 논문에서는 소프트웨어 메트릭스 후보를 선정하여 메트릭스 사이의 코딩 패턴 특성을 조사했다. 코딩 패턴에서 얻어진 정보는 크게 나누어 다음의 2종류가 있다.

- 패턴 자신의 정보 즉, 패턴의 요소 수와 패턴의 인스턴스 수가 이것에 해당한다.
- 패턴의 인스턴스에서 얻어진 정보 즉, 패턴에 해당하는 소스 코드 조각의 위치 등은 여기에 포함된다.

본 논문에서는 이러한 정보를 표현한 지표 값으로 6가지를 선정했다. 패턴 분석을 목적으로 모든 패턴 P를 취합하여 정수 또는 실제 숫자를 반환하는 함수 형태로 정의했다. 다음은 그러한 매트릭스의 정의를 보이며, 패턴 P에 대해 이러한 인스턴스 i는 $i \in P$와 같이 집합 P의 원소로 나타내고, 패턴 P의 인스턴스 수는 |P| 라는 형식으로 기술하는 것으로 한다.

3.1 패턴 길이(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 제거된다.

3.2 패턴의 인스턴스 수(Number of Instances)

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

3.3 제어 요소의 비율(Ratio of Control Elements)

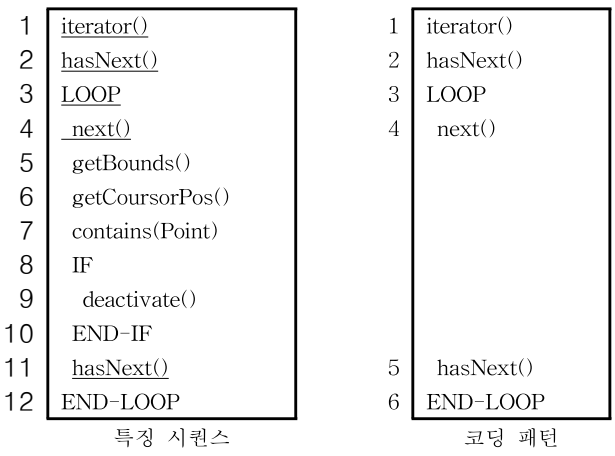
코딩 패턴 P 제어 구조 요소의 비율 RCE(P)는 패턴 P에 포함된 모든 요소에 따른 제어 요소 수를 비율로 계산되는 실수 값이다. RCE(P)의 값이 큰 패턴 P는 메서드 호출을 포함하지 않는 if 문 또는 for 문을 간단한 중첩 관계만을 표현했을 가능성이 높다. 따라서 패턴 마이닝 후, 경험적으로 정한 임계값 $RCE(P) \leq 0.7$ 조건으로 패턴 필터링을 실시한다.

3.4 패턴의 밀도(Density)

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

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

[그림 2]에서 오른쪽에 나타나고 있는 코딩 패턴이, 왼쪽과 같은 인스턴스로서 출현하고 있을 때, 선두 요소 iterator()에서 마지막 요소 END - LOOP까지 패턴에 해당하는 6요소가 출현하고 있다고 가정하면 이 인스턴스에 대한 밀도 $DEN_{inst}(i) = 6/(12-1+1) = 0.5$ 가 된다.



[그림 2] 메트릭스 DEN

패턴 P의 밀도 DEN(P)는 P의 모든 인스턴스의 밀도의 평균으로서 정의되고 있어, 각 인스턴스가 다른 요소를 사이에 포함하지 않는 단일의 코드 조각에 다가가는 만큼, 값이 1에 다가간다.

3.5 반복되지 않는 요소 비율(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 논문에서는 반복 검색은 SEQUITUR 알고리즘[6]을 이용했다. 이 알고리즘은 어느 요소 열이 주어졌을 때, 연속하여 2요소의 집합이 2회 이상 출현하는 경우를 반복으로 검출한다.

3.6 패턴 인스턴스의 분산 : RAD (Radius)

패턴 인스턴스의 분산 RAD(P), 패턴 P의 인스턴스가 배포되는 소스 코드의 범위를 나타내는 정수이다. Java 언어에서 소스 코드 패키지는 계층 구조로 관리하며, 개발 조직의 도메인 이름 및 소프트웨어의 이름, 그리고 소프트웨어에서 서브시스템 이름 등을 이용해 중복되지 않는 명칭을 사용한다[5].

4. 매트릭스를 이용한 코딩 패턴 분석

코딩 패턴의 특성을 조사하기 위하여 3장에서 정의한 6가지 메트릭을 오픈 소스에서 추출한 코딩 패턴에 적용했다. 조사 대상의 소프트웨어의 목록과 실험에 사용된 버전, 소프트웨어의 규모, 실제로 발견된 코딩 패턴의 수를 [표 1]에 나타내었다.

[표 1] 해당 소프트웨어

소프트웨어 이름	버전	규모(LOC)	패턴총계
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 메트릭스 사이의 관련 분석

본 논문에서는 6가지 매트릭스들의 모든 조합에 대해 관련 조사를 실시했다. 여기서는 관련성이 인정되는 매트릭스들의 조합을 말한다.

제어 구조 요소를 포함하는 패턴과 제어 구조 요소가 없는 패턴의 각각의 수를 [표 2]에 나타냈다.

제어 구조 요소를 포함하는 패턴을 포함하지 않는 패턴의 수를 비교하면, 이번 해석한 소프트웨어 모두, 제어 구조 요소를 포함한 패턴의 수가 제어 구조를 포함하지 않는 패턴의 수보다 많이 검출되고 있다.

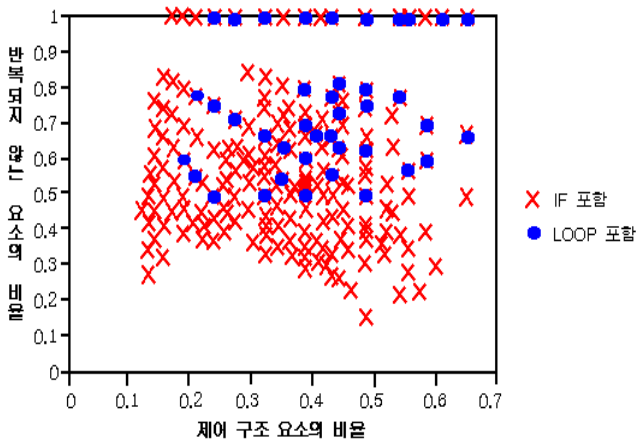
제어 구조 요소가 없는 패턴은 반복되지 않는 요소의 비율에 편향이 없고 넓게 분포하고 있기 때문에, 제어 구조 요소를 포함하는 패턴에만 주목한다.

[표 2] 제어 구조 포함 유무

소프트웨어 이름	제어구조 요소 없음	제어구조 요소 있음
JHotDraw	140	235
jEdit	1212	1690
Apache Tomcat	2489	6293
SableCC	120	330

[그림 3]은 조건부 요소를 포함하는 요소는 전체로 확산되고 있지만, 반복되는 요소는 반복되지 않는 요소의 비율이 1에 가까운 쪽에 치우쳐 분포하고 있다.

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



[그림 3] 제어 구조 요소의 비율에 따른 반복되지 않는 요소 비율 (Apache Tomcat)

4.2.2 반복 출현 위치 관련

반복의 이용과 외관 위치에 대한 분석을 위해서는 코딩 패턴 안에서 반복 사용을 포함하는 패턴을 선별할 필요가 있다.

반복의 이용은 메서드 이름에 "next"와 "hasNext"라는 특징적인 문자열을 가진 메서드 호출을 포함한다. 또한, 반복 사용은 여러 객체에 대해 작업을 반복해야한다. 그래서 코딩 패턴에서 다음 조건을 만족하는 코딩 패턴을, 반복을 사용하는 패턴으로 추출했다.

5. 결론 및 향후연구

코딩 패턴을 유형별로 분류하고, 코딩 패턴을 살피고자 하는 개발자가 필요로 하는 것만을 선별 제시할 필요가 있다. 따라서 본 논문에서는 코딩 패턴의 특성을 측정하기 위한 지표를 제시하고 그 특징간의 관련하여 코딩 패턴의 종류와 관계에 대해 분석했다. 그 결과 패턴의 인스턴스 수, 인스턴스의 분포의 넓이, 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 부품 검색 시스템에서 소프트웨어 부품과 함께 제공하는 것 등을 들 수 있다.

참고문헌

[1] 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.

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

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

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

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

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