

## 코드 가시화부터 모델링 추출을 통한 역공학 적용

권하은\*, 박보경\*, 이근상\*, 박용범\*\*, 김영수\*\*\* 김영철\*

\*홍익대학교 컴퓨터정보통신공학과 소프트웨어공학연구실

\*\*단국대학교 컴퓨터과학과

\*\*\*정보통신산업진흥원 소프트웨어공학센터

e-mail:{kwon, park, yi, bob}@selab.hongik.ac.kr\*,

ybpark@dankook.ac.kr\*\*, ysgold@nipa.kr\*\*\*

## Applying Reverse Engineering through extracting Models from Code Visualization

Haeun Kwon\*, Bokyung Park\*, Keunsang Yi\*,

Young B. Park\*\*, Youngsoo Kim\*\*\*, R. Youngchul Kim\*

\*SE lab, Dept. of Computer Information Communication, Hongik University

\*\*Dept. of Computer Science, Dankook University

\*\*\*National IT Industry Promotion Agency

### 요약

최근 레가시 소프트웨어 기능의 증가와 범위가 넓어져, 결함으로 발생하는 사고의 피해 규모가 증가하고 있다. 그로 인해 소프트웨어의 고품질화가 절대적으로 필요하다. 기존 NIPA의 소프트웨어 가시화는 코드로부터 아키텍처 추출을 중점으로 고려한다. 이로부터 역공학 기법을 통한 객체지향 코드의 정적 분석과 가시화로 모델(클래스 모델, 순차적 모델, 패키지 모델, 그리고 유스 케이스 모델)과 요구사항을 추출하고자 한다. 이에 앞서 기존에 구축한 Tool-chain[6]에서 가시화를 통한 클래스 모델 추출을 먼저 시도한다. 본 논문에서는 객체지향 패러다임에 맞게 수정된 결합도 측정 방법을 제안하고 추출된 UML 클래스 다이어그램에 적용한다. 그 방법은 측정된 결합도를 클래스 간 의존 관계와 비교하고 UML 클래스 다이어그램에 표현하는 것이다. 이를 통해 기존 레가시 소프트웨어의 재개발 과정에서 설계 문서의 추출과 고품질화가 가능하다.

### 1. 서론

최근 소프트웨어 융합 가속화로 인해 기능의 증가와 범위가 넓어지고, 결함으로 발생하는 사고의 피해 규모도 증가하고 있다. 이 때문에 소프트웨어의 고품질화를 위한 요구사항 분석에서 유지보수의 개발 전 과정에 걸친 체계적인 관리와 효율적인 업무수행을 지원해주는 기술의 적용이 요구되고 있다[1].

고품질 소프트웨어를 만드는 방법은 메트릭, 기술과 도구, 자원 관리, 개발 및 테스트 프로세스 관리 등이 있다[2]. 이 중 메트릭은 응집도, 결합도, 복잡도, 라인 수 등의 다방면에서 소프트웨어 품질을 정량적으로 측정하는 기법이다[3]. 이를 기반으로 하는 소프트웨어 가시화 기법은 코드에서 응집도와 결합도를 측정하고 측정 결과를 그래프로 표현한다.

그러나 기존의 응집도와 결합도 측정 방법은 절차지향 패러다임이기 때문에 객체지향 패러다임으로 수정이 필요하다[4]. 이를 해결하고자 객체지향 패러다임에 맞게 수정

된 결합도 측정 방법을 제안한다. 또한 측정한 결합도를 클래스 간 의존 관계와 비교하여 UML 클래스 다이어그램에 가시화한다.\*

본 논문의 구성은 다음과 같다. 2장에서 기존의 소프트웨어 가시화 프로세스에 대해 소개한다. 3장에서는 객체지향 코드에서 결합도를 측정하는 방법과 이를 UML 클래스 다이어그램에 표현하는 방법을 언급한다. 4장에서는 제안한 내용을 실제 소스코드에 적용하고, 5장에서는 결론 및 향후 연구에 대해 언급한다. 이를 통해 소프트웨어 개발 과정에서 설계 문서와 구현된 코드를 대조하고 클래스 간 결합도 관리 방법을 제시한다.

### 2. 관련 연구

#### 2.1. 소프트웨어 가시화

기존의 소프트웨어 개념이란 통상적으로 물리적 형상이나 크기를 갖지 않는 비가시적인 특징이다. 그러나 소프

\* 이 논문은 2014년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업(No. 2012M3C4A7033348)과 2014년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(NRF-2013R1A1A2011601).

트웨어 가시화 기법은 이러한 소프트웨어를 시작적으로 표현하여 쉽게 이해하도록 하는 기법이다. 가시화 하는 세부 대상에 따라 소프트웨어 구조 가시화, 런타임 행동(Runtime behavior) 가시화, 코드 가시화 등으로 구분한다 [5].

본 논문에서 정적 구조의 가시화로, 객체지향 코드를 콜 그레프 형태로 크게 노드(Node)와 엣지(Edge)의 2가지 요소로 구성되는데, 클래스는 노드로, 클래스 간 관계는 엣지로 표현할 수 있다. 예를 들어 하나의 클래스 A에서 다른 클래스 B의 멤버 변수, 메소드, 생성자 등에 접근하거나 호출 관계는 'A→B'와 같이 방향성을 가진 엣지(edge)로 표현 가능하다.

## 2.2. 소프트웨어 가시화 프로세스

기존 연구[6]에서는 오픈 소스 도구들을 활용하여 하나의 도구 체인(Tool-chain)을 구성하고, 자동화된 소프트웨어 가시화 프로세스를 구축하였다. 그림 1과 같이 Tool-chain은 Parser(혹은 Analyzer), Database, View Composer의 3가지 도구로 구성하고 순서대로 소스 분석, DB 저장, 구조 분석, 가시화의 4단계로 이루어진다. 첫 번째 소스 분석 단계에서는 Parser를 통해 객체지향 코드에서 클래스, 메소드, 변수 등의 구성 요소와 요소들 간의 관계를 추출한다. 추출한 구성 요소와 관계 정보는 두 번째 DB 저장 단계에서 데이터베이스에 분류된다. 예를 들어 클래스, 메소드, 변수 별로 테이블을 정의하여 분류하고, 일반화 관계, 연관 관계별로 분류하여 다음 단계인 구조 분석 단계에서 정보를 조회가 용이하다. 세 번째 구조 분석 단계에서는 분류된 정보를 토대로 결합도를 측정한다. 마지막 시각화 단계에서는 분석한 정보로부터 View Composer가 인식 가능한 스크립트를 생성과 이를 가시화 한다. 그림 1은 소프트웨어 가시화를 위한 Tool-chain 구조이다.

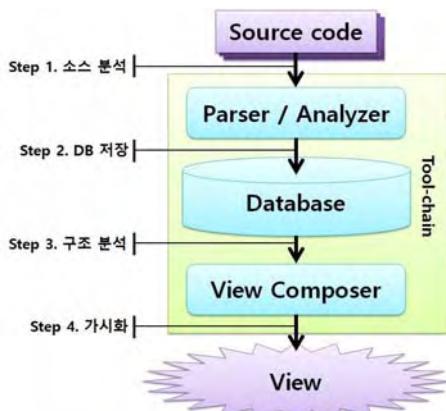


그림 1 소프트웨어 가시화 Tool-chain 구조

## 3. 제안한 모델 추출 및 품질 지표 프로세스

### 3.1. 객체지향 파라다임에서의 결합도 측정

소프트웨어의 고품질화를 위해 소프트웨어 설계 원칙의 '높은 응집도, 낮은 결합도 원리'에 따라 결합도를 정량적으로 측정하고 낮출 필요가 있다. 이를 위해 과거 연구[7]에서 정의한 표 1의 결합도 코드 패턴과 품질 지표 값을 사용하여 자료, 스템프, 제어, 외부, 공유, 내용의 6가지 결합도로 분류하고 값을 측정한다.

표 1 결합도 코드 패턴 및 지표 값

	코드 패턴	점수	가중치 예
자	Circle c = new Circle(); Circle(); c.setRadius(3);	1	0.5
수	int array[5]={1, 2, 3}; MathUtil mu = new MathUtil(); mu.getSum(array);	2	0.5
제	SubPro sp=new SubPro(); ListOb lo=new ListOb(); if(lo.isEmpty()==true) { sp.sub_process1(); } else { sp.sub_process2(); }	3	1
외	Dto dto = new Dto(); send(dto); dto = receive();	4	1.3
함	float circleArea(r) { return r*r*PI.get(); } float shpereArea(r) { return 4*r*r*PI.get(); } class PI() { float PI=3.14; float get(){return PI;} }	5	1.5
내	Circle c = new Circle(); c.r = 5.0; class Circle() { float r; ... }	6	1.7

먼저 코드 패턴을 통해 코드에서 결합도를 검출한다. 소프트웨어 가시화 프로세스의 구조 분석 단계에서 소스 분석 단계와 DB 저장 단계를 통해 분류된 코드 구성 요소를 분석하여 코드 패턴과 일치하는 관계를 찾는다. 예를 들어 자료 결합도를 검출하기 위해서는 하나의 클래스에서 다른 클래스의 메소드를 호출하는 관계를 찾는다. 이때 호출하는 메소드의 인자가 기본 자료형만으로 구성되었는지 판별한다. 이를 만족하면 자료 결합도이며, 메소드 인자로 배열 혹은 클래스를 요구하면 스템프 결합도로 분류 할 수 있다. 다음으로 검출한 결합도에 정량적인 수치 값을 할당한다. 기존 연구[7]와 같이 자료 결합도를 1점으로 각 결합도에 순서대로 6점까지의 점수 값을 정의하였다. 하지만 이를 통해 결합도의 크고 작은은 알 수 있으나 정량적인 수치 값이라 할 수는 없다. 즉 두 클래스 간에 6개

의 자료 결합도가 존재한다고 해서, 1개의 내용 결합도와 동일함을 의미하지는 않기 때문이다. 이를 해결하기 위해 기존 결합도 지표에 추가적으로 가중치(weight) 값을 곱하여 결합도 점수를 측정한다. 가중치 값은 각 결합도 별로 정의하며 기본 값을 1으로 하고, 치명적인 결합도는 1보다 큰 값을 주며, 중요치 않은 결합도에는 0이상 1미만의 값을 부여한다. 예를 들어 객체지향 코드에서 빈번하게 발생하는 자료 결합도나 스템프 결합도의 경우 0.5의 가중치를 주어, 전체 결합도 수치에 미치는 영향을 줄일 수 있다. 반면에 결합도 관점에서 치명적인 내용 결합도에는 1.7의 가중치 값을 주어, 내용 결합도 하나 당  $6 \times 1.7 = 10.2$ 의 점수로 계산할 수 있다. 표 1은 결합도 코드 패턴 및 지표 값을 보여준다.

### 3.2. 역공학 기법을 통한 UML 클래스 다이어그램 추출 및 결합도 적용

코드로부터 UML 클래스 다이어그램 추출 과정은 기존 소프트웨어 가시화 프로세스 내에서 결합도 측정과 함께 수행되며, 클래스 다이어그램을 그리는데 필요한 작업이 추가적으로 행해진다.

먼저 소스 분석 단계에서는 객체지향 코드의 구성 요소와 관계들을 추출함과 동시에 클래스 간 일반화, 연관, 의존 관계 등의 정보를 추출한다. DB 저장 단계에서는 기존과 마찬가지로 추출한 정보를 각각의 테이블에 분류한다. 구조 분석 단계에서는 분류한 의존 관계와 측정된 결합도를 대조한다. 의존 관계는 어떤 클래스 메소드 인자 혹은 메소드 내의 지역 객체를 정의할 때 발생한다. 또한

이는 표 1의 결합도 패턴과 유사하다. 그러므로 결합도와 의존 관계 양쪽에 해당되는 관계를 검출하고, 시각화 단계에서 UML 클래스 다이어그램의 의존 관계에 결합도 값을 나타내도록 한다.

#### 4. 적용 사례

기준에 개발한 “*Use case diagram drawing tool*” 코드를 제안한 프로세스에 적용한다. 이 코드는 자바(Java) 언어로 개발된 객체지향 코드이다. 먼저 소스 분석 단계에서 Parser를 통해 코드로부터 구성 요소와 요소 간 관계를 추출한다. Parser로는 Source Navigator 6.0(SN)[8]을 사용한다. SN을 통해 추출된 정보는 \*.cl, \*.iv, \*.md, \*.in 등의 바이너리 파일로 제공된다. cl 파일은 클래스, iv 파일은 멤버 변수, md 파일은 메소드, in 파일은 일반화(상속) 관계, by 파일 각 요소들 간의 호출 관계에 대한 정보를 담고 있다.

다음으로 DB 저장 단계에서 각 파일에 존재하는 정보를 쉽게 조회하기 위해 데이터베이스에 분류한다. 데이터베이스로는 SQLite[9]를 사용한다. 클래스, 멤버 변수, 메소드, 일반화 관계, 호출 관계 테이블을 정의하고, 앞서 생성된 파일을 읽어 해당하는 테이블에 삽입한다. 각 파일은 바이너리 형태로 존재하기 때문에 SN에서 제공하는 dbdump.exe를 사용해 정보를 추출한다.

구조 분석 단계에서는 분류된 정보를 토대로 결합도를 측정하고 클래스 다이어그램을 만든다. 결합도를 측정하기 위해 이전 단계에서 분류된 호출 관계를 분석하고, 호출하는

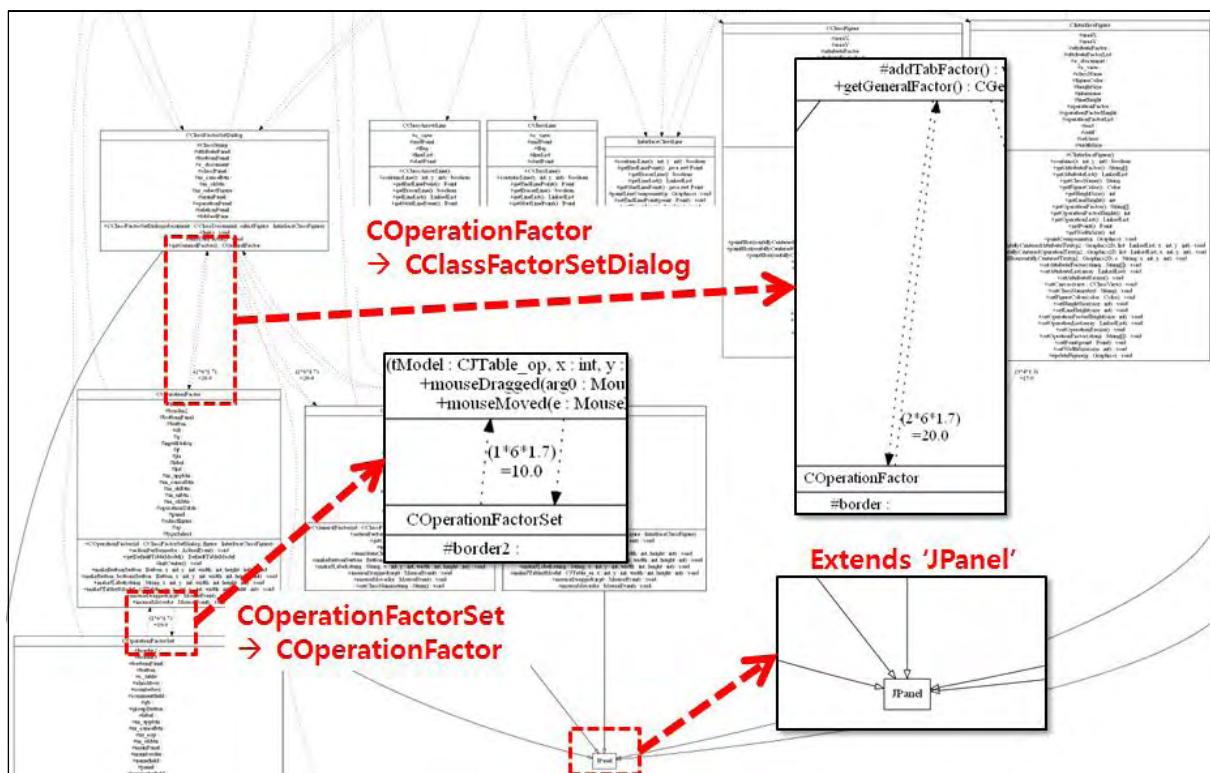


그림 2 UML 클래스 다이어그램에 결합도 적용

는 클래스와 호출되는 메소드 혹은 변수에 관한 정보를 조회한다. 그리고 이를 자료, 스탬프, 제어, 외부, 공통, 내용의 각 결합도 패턴과 대조한다. 결합도 패턴과 일치할 경우 결합도 점수를 측정하며, 호출하는 클래스와 호출되는 클래스의 순서쌍별로 값을 저장한다. 예를 들어 클래스 A, B, C 간에 결합도가 존재할 경우 A→B, A→C, B→A, B→C, C→A, C→B의 순서쌍 각각에 대한 결합도 값을 계산해야 한다.

또한 클래스 디아어그램을 추출위해 데이터베이스에서 일반화 관계와 호출 관계를 조회한다. 클래스 간 상속 관계는 일반화 관계에서, 의존 관계는 호출 관계로부터 추출 할 수 있다. 의존 관계 분석 시, 호출 클래스 간 순서쌍을 결합도 순서쌍과 비교한다. 클래스 순서쌍이 동일하고 동일한 대상(변수 혹은 메소드)에 접근할 경우, 해당 결합도와 의존 관계를 짹짓는다.

마지막 시각화 단계에서는 분석한 클래스 디아어그램 정보를 View Composer인 Graphviz 2.24[10]가 인식 가능한 스크립트로 변환한다. 클래스는 사각형의 노드로, 일반화 관계는 실선으로, 의존 관계는 점선으로 표현된다. 이 때 결합도가 짹지어진 의존 관계는 결합도 값을 함께 표시한다. 또한 클래스의 멤버 변수와 메소드에 관한 정보는 해당 클래스 노드 내부에 나타낸다. 최종적으로 작성된 스크립트를 View Composer에 입력하여 결합도가 적용된 클래스 디아어그램을 표현한다.

그림 2는 시각화된 클래스 디아어그램의 일부분이다. 실선은 일반화 관계를 나타내며, 다수의 클래스가 JPanel 클래스를 상속하고 있음을 나타낸다. 또한 클래스 COperationFactor와 CClassFactorSetDialog 사이의 점선은 두 클래스 간 의존 관계를 나타낸다. 화살표가 CClassFactorSetDialog로 향하는 점선에는 수식이 존재하며, 이는 2개의 내용 결합도(6점)가 존재하고 가중치 값 1.7을 곱한 약 20.0의 결합도 값을 의미한다. 마찬가지로 COperationFactorSet과 COperationFactor 사이에는 1개의 내용 결합도(6점)에 가중치 값 1.7을 곱한 약 10.0의 결합도 값이 존재함을 알 수 있다. 반면에 수식이 없는 점선(의존 관계)은 해당 호출 관계에 알맞은 결합도 패턴이 없음을 의미한다. 이는 각 결합도의 코드 패턴 수를 늘리는 것으로 해결 가능하다. 또한 Parser로 사용한 SN에서 멤버 변수의 타입이나 접근 권한에 대한 정보를 제공하지 않기 때문에, COperationFactorSet 클래스의 멤버 변수 border2의 자료형이 누락된 것을 확인할 수 있다. 이 때문에 클래스 간 연관 관계를 그리는데 어려움이 있으며, 다른 Parser를 사용하거나 SN과 함께 사용하여 부족한 정보를 보완하는 것으로 해결 가능하다.

## 5. 결론

본 논문은 목적은 기존 레가시 소프트웨어를 재개발 과정에서 설계 문서의 추출을 통한 고품질화에 있다. 즉 역공학 기법을 통한 객체지향 코드의 정적 분석과 가시화

로 모델(클래스 모델, 순차적 모델, 패키지 모델, 그리고 유스 케이스 모델)과 요구사항 추출하는 시도이다. 그 결과로 역공학 기법을 통해 개발 도중에도 클래스 디아어그램을 추출 가능하다. 이를 통해 설계대로 코드가 작성되는지 실시간으로 가시적으로 확인 할 수 있다. 또한 클래스 디아어그램에 결합도 값을 통해 정량적 관리 가능하다. 하지만 현재는 제한적으로 품질 척도 중 하나인 결합도만을 측정한다. 그러므로 향후에는 결합도 외의 다양한 품질 척도의 측정 방안을 탐구하고, 이를 검출하기 위한 코드 패턴을 확장하고자 한다.

## 참고문헌

- [1] NIPA 소프트웨어공학센터, “2013 소프트웨어 공학 백서”, 2013. 4.
- [2] Arun Rai, Haidong Song, Marvin Troutt, "Software Quality Assurance: An Analytical Survey and Research Prioritization", Journal of Systems and Software, Volume 40, Issue 1, January 1998
- [3] Ince, D., "Software Metrics: Introduction", Information and Software Technology, Volume 32, Issue 4, May 1990
- [4] 박성희, 홍의석, 우치수, 김태균, “객체 지향 프로그램에서 응집도, 결합도 측정 메트릭 집합”, 한국정보과학회, 제25권, 제12호, 1998. 12
- [5] Thomas Ball, Stephen G. Erik, Bell Laboratories, "Software Visualization in the Large", IEEE Computer Society, Volume 29, Issue 4, April 1996
- [6] 박보경, 권하은, 양효석, 문소영, 김영수, 김영철 “객체 지향 코드의 정적 분석을 위한 Tool-chain화 사례 연구”, 한국정보과학회, 2014. 6
- [7] 권하은, 손현승, 서채연, 김영수, 박병호, 김영철, “기존 모듈 간의 결합도 및 응집도 개념과 객체지향 파라다임과의 관련 비교 연구”, 한국정보과학회, 2014. 6
- [8] emthornber, "Source Navigator NG", <http://sourcenav.sourceforge.net/>
- [9] SQLite, "About SQLite", <http://www.sqlite.org/about.html>
- [10] graphviz, "DOT tutorial and specification", <http://www.graphviz.org/Documentation.php>