

코드 내부 정보의 정규화 기반 효율적인 코드 정적 분석 및 가시화

박찬솔*, 전병국², 김영철¹

¹홍익대학교 소프트웨어융합학과

²강릉원주대학교 컴퓨터공학과

c2193102@g.hongik.ac.kr, jeonbk@gwnu.ac.kr, bob@hongik.ac.kr

Effective code static analysis and visualization based on Normalization of internal code information

Chansol Park¹, Byungkook Jeon², R. Young Chul Kim¹

¹Dept. of Software and Communications Engineering, Hongik University

²Dept. of Com. Sci. & Eng., Gangneun-Wonju National University

요 약

고품질 코드를 위한 정적 분석은 아직도 매우 필요한 영역이며, 또한 코드의 가시화는 개발자들에게 코드의 복잡한 모듈에 대한 가이드에 필요하다. 기존의 코드 가시화는 정적 분석의 코드 내부 정보들을 DB 테이블화 및 품질 지표(CK Metrics, Coupling, # function Calls, Bed smell) 질의어화, 그리고 추출된 정보를 가시화하는 것에만 초점을 두었다. 문제는 코드 내부 정보(Class, method, parameters, etc) 테이블들에 대한 join 연산 시 엄청난 시간과 리소스가 소모된다. 이 문제를 해결하기 위해, 우리는 테이블 설계의 정규화를 제안한다. 또한 필요한 품질 지표의 질의를 통해 코드 내부 정보 추출하여 데이터 및 제어 복잡 모듈을 식별하여 refactoring 를 가이드 한다. 앞으로는 이 부분의 AI learning 을 통해 bad/good program 을 식별을 기대한다.

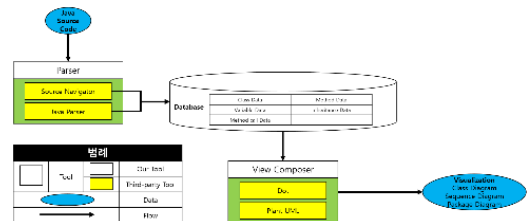
1. 서론

현재 소프트웨어가 다양한 분야에 다양한 소프트웨어의 종류로 많이 사용되고 개발되어야 한다. 이들의 소프트웨어들의 정적/동적 분석을 통한 코드(데이터 및 제어)복잡도를 가시화가 중요하게 여겨진다. 하지만 점차 소프트웨어의 규모가 커짐에 따라 분석해야 할 코드와 분석을 통해 추출한 데이터의 규모도 커져 기존의 우리의 도구를 적용하기에 어려움이 발생한다. 따라서 본 논문에서는 코드 내부의 정보를 추출하여 저장하는 Database(DB)의 테이블 구조를 정규화 하여 코드에 대해 효율적인 정적 분석 및 가시화 기법을 제안한다.

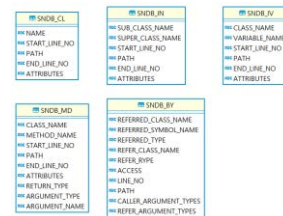
2. 기존 연구

본 연구실에서는 다양한 방식으로 연구가 진행 중이다[1,2,3,4]. 첫 번째 방식은 Open Source 기반 툴 체인화를 통한 코드 정적 분석 연구가 있다[1].

해당 연구에서 사용한 툴 체인은 그림 1 과 같이 Source Navigator 도구를 통해 소스코드를 분석하여 추출된 SNDB 를 기반으로 테이블을 생성한 후 Java Parser 를 통해 SNDB_BY 테이블을 보충하는 구조이다.



(그림 1) 기존 코드 정적 분석 툴 체인 구조.



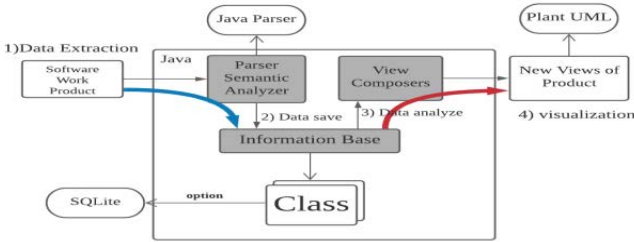
(그림 2) 기존 DB 구조(ER Diagram).

<표 1> 기존 코드 복잡도 추출 쿼리 예시

```
select R.class, R.symbol_name, R.ref_class, R.ref_symbol, R.filename, R.ref_type, R.position
from RefersTo as R LEFT JOIN
MethodImplementation as M
on R.ref_symbol = M.name
where R.ref_type = 'ud' and R.class <> '#'
and M.name IS NULL and R.ref_argument_types = ''
and R.access <> 'p' order by R.class, R.symbol_name, R.position;
```

분석 과정에서 생성된 DB 테이블들은 SNDB 의 과 일 구조와 완전히 동일한 DB 테이블을 생성하였기 때문에 그림 2와 같이 Table들이 정규화 되지 못했다는 문제점이 있다. 이 때문에 표 1과 같이 Join 구문을 통해 데이터를 검색할 때 조건이 복잡하며, 이러한 복잡한 조건을 통해 두 테이블을 Join 하더라도 Dummy data가 생성되거나 혹은 잘못된 데이터가 검색될 수 있다는 심각한 문제점이 존재한다.

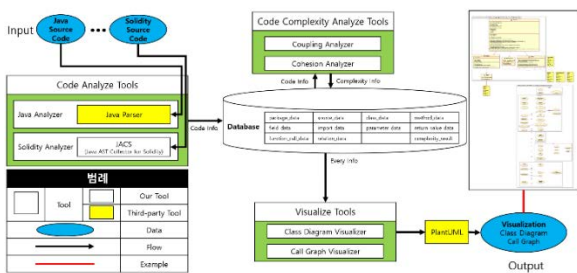
두번째 방식은 역 공학 방식으로 UML 설계 추출 자동화 연구가 있다[2].



(그림 3) 기존 코드 정적 분석 툴 체인 구조.

해당 연구에서 사용한 툴 체인은 그림 3과 같이 Java Parser만을 이용하여 소스코드를 분석하여 추출한 정보를 따로 DB 구조에 저장하지 않고, 정적 분석 및 가시화를 진행하는 구조이다. 이러한 방법의 문제점은 코드에서 추출해낸 모든 정보를 메모리에 저장하기 때문에 대규모의 코드를 추출하기 어려우며, 분석한 정보를 다이어그램 및 그래프 외의 방법으로는 기록하지 않으므로, 지속적인 분석을 통한 시계열 데이터를 형성하는 데에도 어려움이 있다. 또한 JAVA 이외의 코드에 대해서 적용하기에도 어려움이 있다.

3. 코드 내부 정보의 정규화 기반 효율적인 코드 정적 분석 및 가시화



(그림 4) 개선된 코드 정적 분석 툴 체인 구조.

본 논문에서 제안하는 코드 정적 분석 툴 체인의 구조는 그림 4와 같다. 툴 체인은 코드 분석 도구, DB, 코드 복잡도 분석 도구, 가시화 도구 크게 4부분으로 나누어져 있다.

우선 코드 분석 도구는 소스코드를 입력 받아 분석기를 이용하여 코드의 정보를 추출한 후 데이터 베이스에 삽입하는 도구이다.

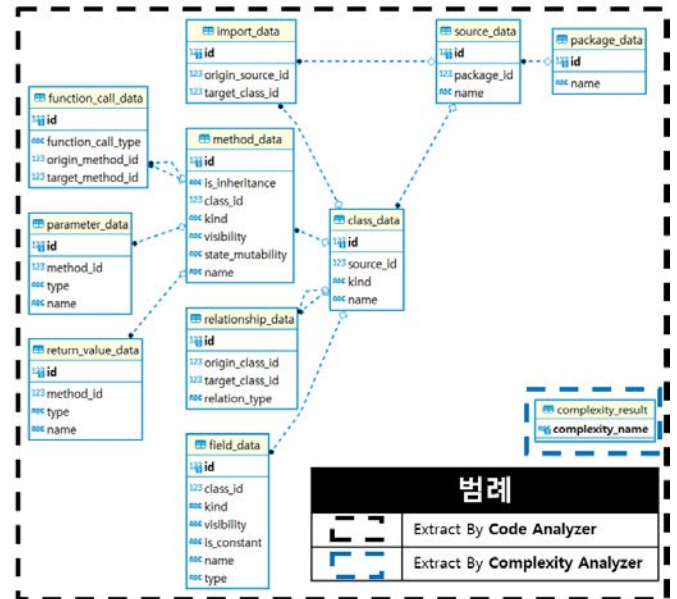
코드 복잡도 분석 도구는 DB로부터 코드 정보를

읽어와 복잡도를 계산한 후 다시 DB에 복잡도를 추가하는 도구이다.

가시화 도구는 여태 분석한 정보 전체를 DB로부터 추출하여 PlantUML 스크립트를 작성하는 도구이다. 이 과정에서 작성된 스크립트는 PlantUML을 통해 다이어그램과 그래프로 그려진다.

마지막으로 DB 구조는 그림 5와 같이 12개의 코드 정보 테이블들과 1개의 복잡도 테이블로 이루어져 있다, 코드 정보 테이블들은 각각 패키지, 소스코드, 클래스 그리고 필드와 메서드의 정보를 저장한다. 이때 메서드의 경우 N개의 매개변수를 가질 수 있고, 언어에 따라 N개의 반환 값을 가질 수 있으므로 이들의 정보를 각각의 테이블로 분리하여 저장한다. 이러한 데이터들을 1차적으로 저장한 후 소스코드 단위에서의 Import, Class 간의 관계, Method 간의 Function Call 정보를 분석하여 저장한다.

개선된 DB에서는 모든 개체에 대해 고유 ID를 부여했으며, 개체 간의 모든 관계를 ID를 통해 연결한다. 따라서 기존의 개체의 이름을 통해서만 분석하는 경우 같은 이름을 가진 개체의 경우 구분하는 것에 어려움이 있었지만, 개선된 DB 구조를 이용하면, 이름을 포함하여 모든 속성이 같은 개체라도 ID를 통해 구분할 수 있다.



(그림 5) 개선된 DB 구조(ER Diagram).

개선된 DB는 JAVA뿐만 아니라 다른 객체 지향 언어에도 사용할 수 있도록 고려하여 설계하였다. 따라서 툴 체인은 Python, C++ 등 JAVA 이외의 객체 지향 언어에 적용할 수 있을 뿐만 아니라 Smart Contract를 구현하는 언어인 Solidity 분석에도 사용될 수 있다.

현재 개발된 툴 체인을 이용해 추출할 수 있는 다이어그램 및 그래프는 Class Diagram과 Call Graph이며, 이중 Class Diagram과 함께 Class에 대해 측정된 복잡

도가 가시화된다. 현재 추출할 수 있는 코드 복잡도는 결합도 관련 복잡도로 CBO, MPC, DAC, RFC 크게 4 종류의 복잡도를 추출하고 있다.

표 2는 CBO를 추출하는 함수 중 일부이다. CBO는 클래스가 결합된 다른 클래스의 개수를 말한다. DB 정규화와 추출한 데이터의 세부적인 분류를 통해 복잡도 추출 Query를 복잡하지 않게 작성해도 되는 것을 볼 수 있다.

<표 2> 코드 복잡도 추출 예시

```

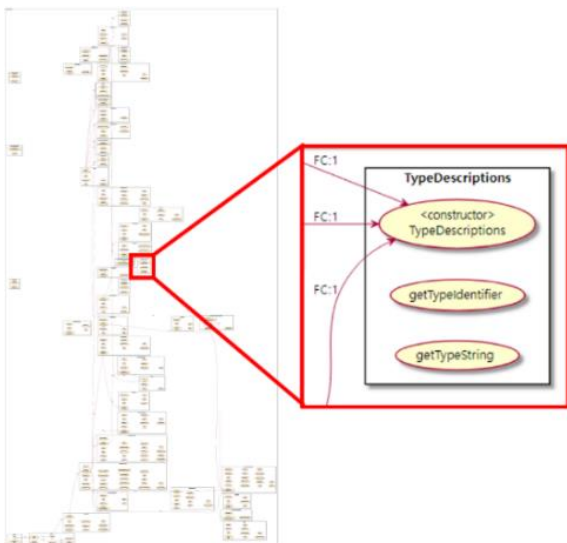
HashSet<Integer> relation = new HashSet<Integer>();
ResultSet targetRS = stat.executeQuery("SELECT * FROM relation_data WHERE origin_class_id = '"+i+"";");
while(targetRS.next()) {
    relation.add(targetRS.getInt("target_class_id"));
}
targetRS.close();
ResultSet originRS = stat.executeQuery("SELECT * FROM relation_data WHERE target_class_id = '"+i+"";");
while(originRS.next()) {
    relation.add(originRS.getInt("origin_class_id"));
}
originRS.close();
cbo.add(relation.size());
    
```

4. 사례

본 논문은 Java에 대한 적용 사례로 직접 개발된 Java AST Collector for Solidity(JACS) 도구에 적용한다. JACS 도구는 solidity의 AST 구조를 Java의 객체 구조에 저장할 수 있도록 하는 도구이다. 또한 Solidity에 대한 적용 사례로 Belp-Gaming 스마트계약에 적용한다.

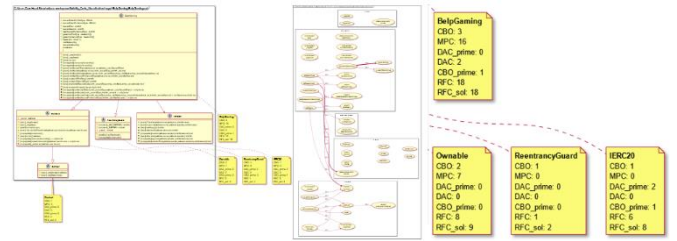


(그림 6) 추출된 Class Diagram.



(그림 7) 추출된 Call Graph.

그림 6은 틀 체인을 통해 추출한 Java의 Class Diagram과 복잡도이며, 그림 7은 Call Graph이다.



(그림 8) 소스코드에 대해 추출된 Class Diagram과 Call Graph.

그림 8은 틀 체인을 통해 추출한 Solidity의 Class Diagram 및 Call Graph이다.

그림 6, 8 처럼 Java 및 Solidity에 대해 CBO, MPC, DAC, RFC와 해당 복잡도에 대해서 일부 수정된 복잡도인 DFC prime, CBO prime 등을 식별하여, 가시화를 통해 코드 내부의 복잡도를 가이드가 가능하다.

그림 7, 8 처럼 Java 및 Solidity에 대해 Method 간의 Call 관계 및 Call 횟수를 화살표의 두께 및 숫자로 표기했다. 추출된 지표 값을 기반으로 지정된 수치를 초과한 코드 영역을 식별하여 refactoring하고자 한다.

5. 결론

본 논문에서는 정적 분석 틀 체인의 DB 구조를 정규화 하여 코드에 대해 정적 분석과 가시화를 효율적으로 하였다. 추후에 코드 내부 정보의 DB 테이블의 정규화로 역 공학 기반 Function Point 값에 대한 검증이 가능하다. 또한 AI learning을 통해 정확히 코드 품질화를 연구를 진행할 예정이다.

ACKNOWLEDGEMENT

이 논문은 교육부 및 한국연구재단의 4단계 두뇌한국 21 사업의 지원(F21YY8102068)과 2022년도 정부(교육부)의 재원으로 한국연구재단의 지원(No. 2021R111A305040711, No. 2021R111A1A01044060)을 받아 수행된 연구임.

참고문헌

- [1] 강건희, et al. "Open Source 기반 틀 체인화를 통한 코드 정적 분석 연구." 정보과학회 컴퓨팅의 실제 논문지, 21.2, 148-153, 2015.
- [2] Jung, Se Jun, et al. "Automatic UML Design Extraction with Software Visualization based on Reverse Engineering." International journal of advanced smart convergence, 10.3, 89-96, 2021.
- [3] Kwon, Haeun, and R. Young Chul Kim. "Extracting Use Case Design Mechanisms via Programming based on Reverse Engineering." International Journal of Applied Engineering Research 10.90, 503-505, 2015.
- [4] Park, Bo Kyung, et al. "Code Visualization for Performance Improvement of Java Code for Controlling Smart Traffic System in the Smart City." Applied Sciences 10.8, 2020.