한국정보과학회

제 22 권 제 1 호
Vol. 22 No. 1

SOFTWARE ENGINEERING SOCIETY

2020
제 22 회 한국 소프트웨어공학 학술대회 논문집

Proceedings of the 22nd Korea Conference on Software Engineering (KCSE 2020)

- 일시: 2020년 2월 3일(월) ~ 2월 5일(수)
- 장소: 강원도 평창 한화리조트(휘닉스파크점)

주최: 한국정보과학회, 한국정보처리학회
주관: 한국정보과학회 소프트웨어공학 소사이어티
한국정보처리학회 소프트웨어공학연구회

후원: ETRI 한국전자통신연구원, SOLUTIONLINK, ㈜비트컴퓨터,
㈜모아소프트, ㈜이에스지,
한국소프트웨어기술진흥협회(KOSTA), T3Q㈜,
㈜다한테크, 슈어소프트테크㈜, ㈜온페이스,
STA 테스팅컨설팅㈜, TTA 소프트웨어시험인증연구소,
㈜에스피아이디, 신뢰적지능형 CPS 연구단

2 월 4 일 (화)

시 간	행 사 내 용			
	논문 발표 B			
	B1: SW 안전 좌장: 서영석(영남대) 장소: 세미나실 1	**B2: 스마트 시스템 I** 좌장: 이지현(전북대) 장소: 세미나실 2	**B3: 인공지능 II (AI for SE)** 좌장: 남재창(한동대) 장소: 세미나실 3	**B4: SW 분석** 좌장: 홍장의(충북대) 장소: 그랜드홀 2
11:10-12:30 (80 분)	소프트웨어 위험원 분석을 위한 상황 분류 기반의 조합을 통한 STPA 문맥표 최적화 [일반논문] 양현수, 권기현(경기대) 결함 트리와 마코프 모델을 이용한 안전 기능의 정량적 검증 **[최우수 일반논문]** Ngoc-Tung La, 김소연, 권기현(경기대) 기능 안전 표준 기반의 무기체계 소프트웨어 개발 및 관리 매뉴얼 문제점 분석 및 개선 방안 제시 **[우수 산업체 논문]** 김태현, 박다운, 백옥현(국방과학연구소) STPA 를 이용한 군집 운행 시스템의 안전성 분석 사례 연구 [단편논문] 김의섭, 유준범(건국대)	Effect-driven Dynamic Selection of Physical Media for Visual IoT Services using Reinforcement Learning **[초청논문 ICWS'19]** K. Baek, I-Y. Ko 데이터베이스 서버 그룹 관리를 위한 EMS [산업체 논문] 김효일(KT), 백종문(KAIST) 트리거-액션 기반 서비스 매쉬업의 신뢰도를 개선하기 위한 전제 상태 및 목표 상태 검증 방법 [단편논문] 김상훈, 고인영(KAIST) BERT 기반의 SNS 메시지 불안 분류기 및 사회적 불안 분포 시각화 시스템 [단편논문] 송지수, 최용석(한양대)	소프트웨어 결함 예측의 조선해양/해상운송 산업 적용 사례 연구 **[우수 일반논문]** 강종구(KAIST), 류덕산(전북대), 백종문(KAIST) 순환 인공 신경망을 활용한 코드 변경 추천 시스템의 학습 시간 단축 방법 연구 **[우수 일반논문]** 배병일, 강성원(KAIST), 이선아(경상대) 딥러닝 기반 버그 추적 기법의 OOV 단어 처리를 위한 형태 정보와 문맥 정보 통합 [단편논문] 김영경, 김미수, 이은석(성균관대) 이슈 보고서와 사용자 매뉴얼 간의 추적성 수립을 위한 머신러닝 기법들의 비교 [단편논문] 조희태, 박도제, 이선아(경상대)	Precise Learn-to-Rank Fault Localization using Dynamic and Static Features of Target Programs **[초청논문 TOSEM]** Y. Kim, S. Mun, S. Yoo, M. Kim 행위 모델 기반 RESTful 웹 어플리케이션 결함 위치 추정 **[우수 일반논문]** 장종인, 이낙원(KAIST), 류덕산(전북대), 백종문(KAIST) 조치 가능한 버그 예측을 위한 유사 패치 추천 **[우수 단편 논문]** 신지호, 남재창(한동대) Python 코드를 위한 정적 분석기 개발 [단편 논문] 홍제성, 박보경, 장우성, 이원영, 정세준, 김영철(홍익대)
12:30-13:40	**중식**			

	논문 발표 C			
	C1: 학부생 논문 I 좌장: 서영석(영남대) 장소: 세미나실 1	**C2: 학부생 논문 II** 좌장: 홍신(한동대) 장소: 세미나실 2	**C3: 블록체인** 좌장: 김순태(전북대) 장소: 세미나실 3	**C4: SW 테스팅 I** 좌장: 백종문(KAIST) 장소: 그랜드홀 2
13:40-15:40 (120 분)	CNN 모델 평가를 위한 이미지 데이터 증강 도구 개발 **[우수 학부생 논문]** 최영원, 이영우, 채흥석(부산대) 소프트웨어 소스 코드의 다중 카테고리 분류를 위한 딥러닝 기반 기법 김민하, 심규진, 이찬근(중앙대) YOLO 모델을 활용한 영상 내의 유해 정보 검출 및 필터링 시스템 개발 조성윤, 권용준, 김채록, 최지원, 김석호, 최나람, 김민석, 정순기(경북대) 텍스트 마이닝과 TF-IDF 유사도 가중치를 이용한 연관 아이템 발견 김민정, 김주창, 박성수, 신동훈(경기대), 정호일(대림대), 정경용(경기대) 문서 유형에 따른 분류를 이용한 마이닝 기반 토픽 추출 구병국, 최소영, 강지수, 백지원(경기대), 박찬홍(상지대), 정경용(경기대) AI Impact on Software Engineering [후원 산업체 발표] 김동영(한국 IBM)	블록체인 기반 CCTV 영상정보 무결성 지원 방안 강민구, 홍준기, 송정한, 김태영, 김순태(전북대) STEAM 교육과 게임을 도입한 EPL 기본 설계 **[우수 학부생 논문]** 이정희, 김민우, 조성휘, 김정아(가톨릭관동대) 도커시스템에서 사용 가능한 모니터링 도구 비교 분석 정채은, 박용범(단국대) 미들웨어 기반 빅데이터 시각화 시스템 아키텍처 설계 백재형, 송진범, 서상원, 남재창(한동대) AI 와 SE 를 활용한 수화 번역 프로젝트 이유렬, 김은진, 이혁진, 이선아(경상대) CCTV 를 이용한 실시간 폭행 및 난동행위 인식 시스템 개발 권용휘, 전승원, 배재빈, 김성윤, 김석호, 정순기(경북대)	이더리움 스마트 컨트랙트 상태 모니터링 시스템의 설계 및 구현 **[초청논문 JSES]** 홍준기, 김순태, 류덕산 실시간 인식 기반의 제품의 표시 정보 추출을 위한 모바일 어플리케이션 설계 및 구현 **[초청논문 JSES]** 민경식, 최지수, 이철훈, 정동주, 이병정 계약 관련 음성 녹취 정보의 신뢰성 있는 저장과 체계적 관리를 위한 블록체인 기반 시스템 구성 방식 [단편 논문] 김인근, 서중원, Alshiri Saad, 장민오, 이유정, 박수용(서강대) 그룹 리더 선출을 통한 비트코인 병렬 트랜잭션 처리 [단편 논문] 이도현, 송아람, 이선재, 박우람, 박수용(서강대) 블록체인 코드의 복잡도 측정을 위한 코드 가시화 [단편 논문] 안현식(홍익대), 박지훈, 김기두(TTA), 박보경, 조재형, Md Ibrahim Khalil(홍익대), 김동호(데이터메트릭스), 김영철(홍익대) 블록체인 기반 실시간 수하물 트레킹 시스템 [단편 논문] 이열국, 이동현, 박수용(서강대)	Concolic Testing for High Test Coverage and Reduced Human Effort in Automotive Industry **[초청논문 ICSE '19]** Y. Kim, D. Lee, J. Baek, M. Kim Javadoc 대상 테스트 요구사항 추출 기법의 정확도 평가 [단편 논문] 김지웅, 홍신(한동대) Enhancing Patch Validation in APR by Introducing Test Case Prioritization and Sampling [단편 논문] Y. Venugopal, Q-N. Phung, E. Lee(성균관대) 무인이동체 소프트웨어 시험 평가 기준 및 절차 연구 [단편 논문] 장정훈(모아소프트) DO-178C 기반의 항공 SW Safety 검증 프로세스 [후원 산업체 발표] 강유선(모아소프트)
15:40-16:00	**휴식**			
	기조강연 II 장소: 그랜드홀 2			사회: 이병정 대회장 (서울시립대)
16:00-16:50	인공지능과 인간의 공존, 그리고 포스트 휴머니즘 이중원 교수 (서울시립대, 한국철학회 차기회장)			
	기조강연 III 장소: 그랜드홀 2			사회: 김정아 대회장 (가톨릭관동대)
16:50-17:40	Build AI-Powered Applications through Data driven Innovation 최창남 부문장 (한국 오라클)			

Python 코드를 위한 정적 분석기 개발

홍제성[1], 박보경[1*], 장우성[1**], 이원영[1***], 정세준[1+], 김영철[2]
홍익대학교 소프트웨어공학연구실
{hong[1], park[1*], jang[1**], leewy[1***], jeong[1+] , bob[2]}@selab.hongik.ac.kr

Development of Static Analysis Tool for Python Source Code

Je Seong Hong[1], Bokyung Park[1*], Woo sung Jang[1**], Won Young Lee[1***], SeJun Jung[1+], R. Young Chul Kim[2]
SE Lab, Dept. of Software & Communications Engineering, Hongik University[1, 2]

요 약

앞으로의 AI 및 빅데이터 관련 소프트웨어의 오류, 오작동으로 인해 큰 인명 피해 등의 손실 발생이 가능하다. 특히 이들의 소프트웨어 코드의 크기와 내부 복잡도가 품질에 비중이 커질 것이다. 그러나 기존 절차식 언어 초점인 McCabe의 복잡도로는 새로운 Python 코드 복잡도 측정은 매우 부족한 실정입니다. 특히 거대한 소프트웨어의 비가시성은 내부 복잡성을 알기 어렵다. 또한 기존의 Python 코드 정적 분석 도구는 가시화 기능이 부족하다. 이를 위해, 소프트웨어의 내부를 가시화하고, 소프트웨어의 품질을 관리하고, 유지보수에 용이하기 위한 정석 분석기를 제안하다. 이를 기반으로 소프트웨어의 내부 구조 관계와 클래스, 모듈 내부의 복잡도 메트릭스 그리고 클래스 내부의 응집도를 측정하여 가시화를 함으로써 소프트웨어의 품질 향상에 기여한다.

1. 서 론

최근 4차 산업혁명과 더불어, 소프트웨어는 다양한 곳에 적용되며 그 비중 또한 매우 커지고 있다. 커진 소프트웨어의 비중만큼 소프트웨어의 오작동, 오류는 막대한 손실을 일으킨다. 국내 소프트웨어 산업은 열악한 개발환경, 인력, 비용 등의 문제로 제대로 된 소프트웨어 유지보수, 품질관리가 어렵다. 또한, 소프트웨어의 비가시성의 특징으로 복잡하고 크기가 큰 소스코드 내부를 쉽게 파악하기 어렵다. 이러한 문제를 해결하기 위해서, 소스코드의 내부를 가시화한다. 가시화한 결과를 토대로 소스코드의 수정, 관리에 도움을 줄 것으로 기대한다.

본 논문은 AI, Big Data에 주로 쓰이며, 계속해서 사용자가 많아지고 있는 Python 프로그래밍 언어를 대상으로 정적 분석 및 가시화를 한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로 소프트웨어 정적 분석 메트릭스, 소프트웨어 응집도, Python AST를 설명한다. 3장에서는 기존의 Python 정적 분석 도구와 본 논문에서의 분석 도구를 비교 및 설명을 한다. 4장에서는 도구 적용 사례를 설명한다. 마지막으로 5장에서는 결론 및 향후 연구를 설명한다.

2. 관련 연구

2.1 소프트웨어 정적 분석 메트릭스

소프트웨어의 코드 메트릭스는 소프트웨어 프로그램 코드의 복잡성, 유지보수성, 비용 등을 정량적으로 파악한다. 이를 통해, 잠재적 위험을 식별하고, 품질을 측정할 수 있다. 본 논문에서 활용한 정적 메트릭스는 다음과 같다.

2.1.1 McCabe's Cyclomatic Complexity

McCabe's Cyclomatic Complexity[1]는 소프트웨어 프로그램 코드의 복잡도를 측정하기 위해 프로그램의 제어 흐름을 기반으로 한다. 이 제어 흐름은 그래프(Graph: G), 선(Edge: e), 마디(Node: n)로 표현할 수 있다. 아래 그림[1]의 공식은 Cyclomatic Complexity의 공식이다.

Cyclomatic Complexity(CC)

$$CC = V(G) = e - n + 2p$$

- e : Graph 내의 Edge 수
- n : Graph 내의 Node 수
- p : 연결되어 있지 않은 Graph 부분 수

그림 1 Cyclomatic Complexity 공식

복잡도를 구하는 공식을 통해 계산된 복잡도 수치를 아래의 표[1]과 같이 A부터 F 등급으로 구분한다.

표 1 Cyclomatic Complexity's Rank

Cyclomatic Complexity (CC)	*Rank*
1 – 5	A
6 – 10	B
11 – 20	C
21 – 30	D
31 – 40	E
CC > 40	F

복잡도가 낮을수록 프로그램의 구조가 안정적이며, 높은 테스트 가능성과 낮은 위험도와 비용이 든다. 이와 반대로 복잡도가 높을수록 비 구조적임을 알 수 있다. 이는 낮은 테스트 가능성, 높은 위험도 및 비용이 든다.

2.1.2 Raw Metrics

Raw Metrics[2]는 원천 코드의 라인을 분석하는 것으로 아래 표[2]의 항목을 측정한다.

표 2 Raw Metrics 측정 항목 표

측정 항목	*내용*
LOC (Lines of Code)	코드 라인 수의 총합
LLOC (Logical Lines of Code)	코드의 논리적 라인 수
SLOC (Source Lines of Code)	코드의 소스 라인 수
Comments	주석의 라인 수
Multi	여러 줄로 나타낸 문자열 수
Blank	공백 라인 수
Single Comments	주석만 있는 라인의 수

2.1.3 Halstead Metrics

Halstead Metrics[3]는 프로그램의 코드가 만들어져 완성된 후 측정하는 방법이다. 코드 내부의 연산자와 피연산자의 수를 기반하여 프로그램의 논리의 규모를 산정하는 메트릭스이다. 아래의 표[3]는 Halstead Metrics의 연산자 추출 항목을 나타낸 것이다.

표 3 Halstead Metrics 연산자 추출 항목 표

추출 항목	*내용*
n_1	코드에서의 연산자 개수
n_2	코드에서의 피연산자 개수
N_1	연산자의 총 발생 빈도수
N_2	피연산자의 총 발생 빈도수

위 표[3]에 나타난 항목들의 관계를 통해 아래의 표[4]와 같이 프로그램의 복잡도를 측정한다.

표 4 Halstead Metrics 측정 항목 표

측정 항목	*내용*
Program vocabulary	$n = n_1 + n_2$
Program Length	$N = N_1 + N_2$
Calculated program length	$\hat{N} = n_1 log n_1 + n_2 log n_2$
Volume	$\mathrm{V} = N log_2 n$
Difficulty	$\mathrm{D} = \frac{n_1}{2} \times \frac{N_2}{n_2}$
Effort	$\mathrm{E} = \mathrm{D} \times \mathrm{V}$
Time required to program	$\mathrm{T} = \frac{E}{18}$
Number of delivered bugs	$\mathrm{B} = \frac{V}{3000}$

2.2 소프트웨어 응집도

소프트웨어 응집도(cohesion)는 모듈안의 요소들이 서로 연관되어 있는 정도로 모듈이 독립적인 기능으로 되어 있는 정도를 나타낸 것이다. 모듈의 응집도가 높을수록 독립적인 모듈이라 할 수 있다[4]. 여러 기능을 수행하는 경우 기능이 제대로 분산되지 않아 재사용하기 어렵다[5]. 따라서 소프트웨어는 모듈의 응집도가 높을수록 좋다.

본 논문에서 측정하는 응집도는 아래 그림[2]과 같이 Python 코드의 객체 설계 관점에서의 응집도를 측정한다.

```
class MyClass(): //클래스 MyClass

    class_var = 0 //클래스 멤버 변수 class_var

    def func1(self): //메소드 func1
        self.instance_var = 0 //인스턴스 변수 instance_var
        return class_var+self.instance_var
        //멤버 변수, 인스턴스 변수 사용

    def func2(self): //메소드 func2
        return self.instance_var+1
        //인스턴스 변수 사용
```

그림 2 클래스와 내부 변수와의 응집도

클래스의 메소드들이 클래스의 멤버 변수, 인스턴스 변수를 하나 이상 사용하는 것에 대한 응집도를 측정한다. 이는 클래스에 속한 메소드와 변수가 서로 묶여 의존하는 구조이기 때문이다.

아래의 표[5]는 이외의 응집도의 종류와 정의를 나타낸 것이다.

표 5 응집도의 종류와 정의

응집도	*정의*
기능적 응집도	모듈 내부의 모든 기능 요소들이 단일 문제와 연관되어 수행될 경우의 응집도
순차적 응집도	모듈 내의 하나의 활동으로부터 나온 출력 데이터를 그 다음 활동의 입력 데이터로 사용할 경우의 응집도
교환(통신)적 응집도	동일한 입력과 출력을 사용하여 서로 다른 기능을 수행하는 요소들이 모인 경우의 응집도
절차적 응집도	다수의 관련 기능을 순차적으로 수행하는 경우의 응집도
시간적 응집도	특정한 시간대에 수행되는 몇 개의 기능들이 모여 모듈화 된 경우의 응집도
논리적 응집도	모듈의 요소가 논리적으로 관계가 있는 경우의 응집도
우연적 응집도	서로 관계가 없는 요소가 모여 모듈을 구성하는 경우의 응집도

2.3 Python 추상 구문 트리(Abstract Syntax Tree)

추상 구문 트리(Abstract Syntax Tree, AST)는 프로그램의 소스코드의 구조를 트리형태의 자료구조로 나타낸 것이다. 이 추상 구문 트리로 심볼 테이블을 구성하여 바이트코드를 생성할 수 있다[6]. Python 언어에는 이 추상화 구문 트리를 생성하는 ast 모듈이 내재되어 있다. ast 모듈의 사용 예시로, 아래 그림[3]의 Python 소스 코드를 ast 모듈로 파싱을 하면 다음과 같이 구분되어 파싱된다.

```
def my_func(a, b):
    """
    Function example
    """
    return a + b
```

그림 3 Python 예시 소스 코드

```
Module(body=[FunctionDef(name='my_func',
args=arguments(args=[arg(arg='a',
annotation=None), arg(arg='b', annotation=None)],
vararg=None, kwonlyargs=[], kw_defaults=[],
kwarg=None, defaults=[]),
body=[Return(value=BinOp(left=Name(id='a',
ctx=Load()), op=Add(), right=Name(id='b',
ctx=Load())))], decorator_list=[], returns=None)])
```

아래 그림[4]는 Python 소스코드를 ast 모듈을 사용해 파싱한 추상 구문 트리를 그림으로 표현한 것이다.

ast 모듈을 사용하여 코드의 정적 분석에 필요한 정보를 분석하여 원하는 정보를 추출을 통해 분석기를 구현할 수 있다.

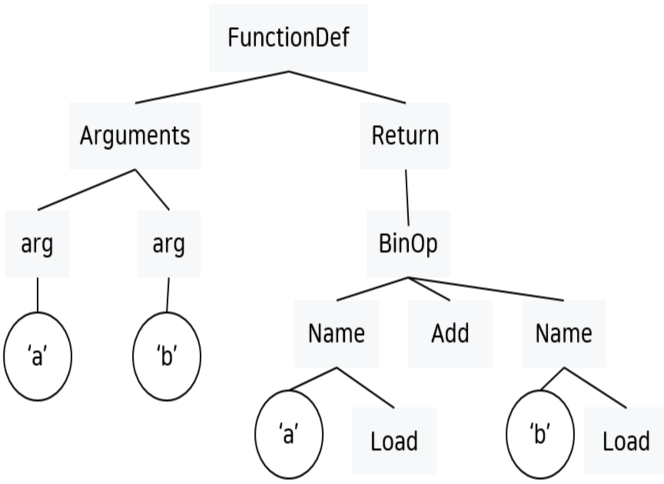

그림 4 추상 구문 트리의 결과 그림

3. 본론

3.1 기존의 Python 코드 정적 분석기와의 비교

기존에 있던 Python 코드 정적 분석기는 Source Meter와 Sonarqube 가 있다. 이 분석기들은 C/C++, Java, C#, Python 등의 프로그래밍 언어의 정적 소스 코드 분석을 위한 도구이다. 정적 분석에 사용되는 도구인 Cppcheck, PMD, FindBugs, FxCop, Pylint 등의 도구들을 포함한 기능을 수행한다. 이와 함께, Python 코드의 각 모듈 사이의 호출관계, 호출 횟수, 소요시간, 메모리 사용량을 그래프화 하여 그리는 패키지인 python call graph가 있다.

Source meter와 Sonarqube는 Python 코딩 룰 체크, 코드 복잡도의 정도를 수치화 하여 보여주는 것에 그친다. python call graph는 모듈 간의 호출 관계, 횟수, 소요 시간, 메모리 사용량 정보를 그래프화 하여 보여주는 데에 그친다. 그러나, 본 논문의 python 코드 정적 분석기는 기존의 분석 도구들과는 다르게 모듈 호출 관계, 횟수, 호출에 필요한 파라미터 개수와 각각의 복잡도 메트릭스(McCabe's complexity, Raw metrics, Halstead metrics) 계산 정보, 응집도 정보를 그래프에 나타낸다. 그려낸 그래프에서 모듈 간의 호출 횟수가 설정한 기준보다 많이 호출되는 호출 관계에는 빨간색 굵은 실선으로 표현하여 가시화한다. 그리고, 2장에서 언급한 복잡도의 기준을 알파벳으로 등급을 나누었듯, 코드의 복잡도가 C등급 아래인 구성 요소들을 빨간색으로 표현하여 복잡도가 높다는 것을 가시화한다.

3.2 Python 코드 정적 분석기

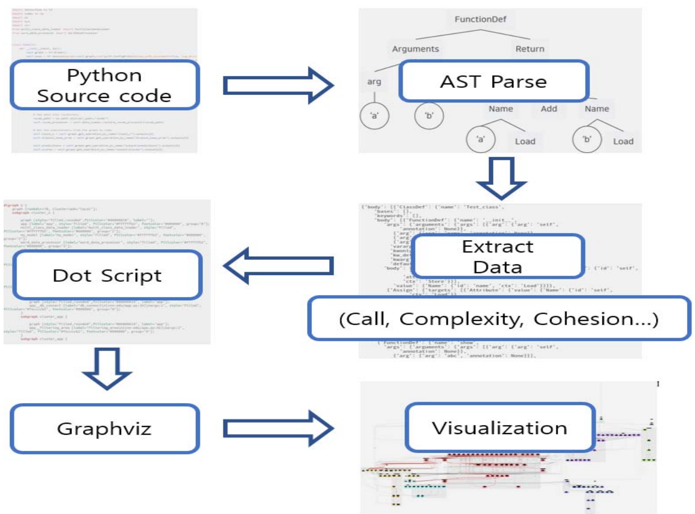

그림 5 Python 코드 정적 분석기 가시화 과정

그림[5]는 Python 코드를 정적 분석하여 가시화하는 과정을 나타낸 것이다. 1)먼저 분석하고자 하는 Python 코드 파일들을 입력한다. 2)입력된 코드들을 AST 구조로 구성하여 파싱을 한다. 파싱된 정보들은 2.3절에 기술된 것과 같이 ast 모듈을 사용하여 값을 반환 받는다. 모듈, 클래스, 함수 내부의 구조를 추출할 수 있다. 3)ast 구조로 파싱된 결과를 통해 호출 정보,

복잡도 정보, 응집도 정보 등 필요한 정보를 추출한다. Cyclomatic 복잡도의 정보는 파싱한 ast 구조에서 분기점이 되는 항목을 찾는다. 기본적인 복잡도 수치는 1이다. 복잡도를 만드는 추출 항목은 다음과 같다. if, elif, for, while, except, with, assert, Comprehension, Boolean Operator. 이러한 구조가 발견되면 Cyclomatic 복잡도를 1씩 증가하는 계산을 한다. Raw, Halstead 복잡도 추출도 역시 ast 파싱 결과를 통해 2장에 언급된 측정 항목 표의 수식으로 계산하여 프로그램의 라인 수, 크기 등의 항목을 추출한다.

응집도 항목은 클래스와 그 내부의 변수 사이의 응집도를 계산한다. 클래스에 생성된 멤버 변수, 인스턴스 변수의 개수를 계산하며, 각 함수에서 몇 개의 클래스에서 생성된 변수를 사용하는지를 계산한다. 4)추출된 정보를 사용해 그래프를 그리기 위한 Dot Script를 생성한다. 5)생성된 Dot Script를 Graphviz[7]에 입력하여 그래프를 생성한다. 6)코드 정적 분석을 위한 가시화 결과를 나타낸다.

4. Python 코드 정적 분석기 활용 사례

분석으로 입력된 프로그램은 Text-CNN을 활용한 교육 프로그램 추천 프로그램[8]의 한 부분이다. 각 클래스, 함수들은 타원형으로 표현된다. 각각의 그룹화를 위해 클래스와 그 내부에 생성된 메소드는 같은 계열의 색으로 표현되며, 채도가 더 진하게 표현된다. 클래스, 함수의 이름과 호출에 필요한 파라미터 개수, 각 모듈, 클래스, 클래스 내부에서의 호출 관계를 실선으로 표현한다. 호출 횟수는 실선위에 숫자로 표기가 된다.

```
def clean_dir(loc):          1             2
    all_subdirs = ... for d in ... if ...
    ...

  3 if dir_path != un_useable:
        ...
  4 if len(file_list) == 0:
        ...
```

그림 7 clean_dir 함수의 구조

그림[7]은 clean_dir 함수의 구조이다. 이 함수의 Cyclomatic 복잡도는 for문 1개 if문 3개의 구조로 이루어져 5점의 값인 A등급이 표시된다. 6)그러나 전달받은 값을 특정 조건에 맞게 필터링하는 filtering_cnn_rate 함수는 Cyclomatic 복잡도가 높게 측정된다.

그림[8]은 filtering_cnn_rate 함수의 구조를 나타낸 것이다. 이 함수의 제어 흐름 구조는 총 10개가 추출되어 11점의 C등급으로 빨간 칸으로 표시된다. 이러한 가시화 결과를 통해 프로그램의 어떤 함수에서 복잡도를 높이고 있는지 알 수 있다.

```
def filtering_cnn_rate(items,cnn_result):
  1 for ...
     2 if ...
        3 if ...
  4 if ...
     5 for ...
          if 6 ...
     7 for ...
          if 8 ...
     9 for ...
          if 10 ...
```

그림 8 filtering_cnn_rate 함수의 구조

5. 결론 및 향후 연구

본 논문은 Python 언어로 된 프로그램의 정적 분석을 통한 가시화를 수행한다. 소프트웨어의 품질을 높이기 위해 모듈 간, 모듈 내부에서의 호출, 복잡도, 응집도를 가시화한다. 분석한 가시화 결과를 사용자에게 제공함으로써 프로그램의 구조를 알 수 있으며, 소스 코드의 어느 부분을 수정하여 복잡도를 낮추고 품질을 더 높일 수 있을지의 척도가 될 수 있다. 향후에는 더 많은 응집도와 결합도를 추가적으로 확장하여 연구할 예정이다.

Acknowledge

본 논문은 2019년도 산업통상자원부의 ‘창의산업융합 특성화 인재양성사업’(과제번호 N0000717)과 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(NRF-2017R1D1A3B03035421)

6. 참고 문헌

[1] T.J.McCabe, “A Complexity Measure”, IEEE Trans. SE, Vol.2, No.6, pp.308-320, Dec. 1976.

[2] Radon, https://radon.readthedocs.io

[3] Halstead, Maurice H. (Maurice Howard) Elements of software science. Elsevier, New York, 1977.

[4] Chidamber Shyam and kemerer Chris, “A metrics suite for object oriented design”, IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp476-493, 1994.

[5] E.-Y. Byun, H.-S. Son, S.-Y. Moon, W.-S. Jang, B.-K. Park, and R. Y. Kim, “정적/동적 분석 기반의 재사용 메트릭과 가시화 구축”, 한국정보처리학회, Vol. 24, No. 1, pp.621-624, 2017.

[6] Python ast, https://docs.python.org/3/library/ast.html

[7] Graphviz, https://www.graphviz.org/

[8] 홍제성, 박보경, 곽제일, 손현승, 김영철, “Text-CNN 알고리즘 적용한 교육장터 플랫폼 기반 맞춤형 교육 컨텐츠 추천 메커니즘 개발”, 한국정보처리학회, Vol. 26, No. 2, pp. 965-967, 2019.

Electronics & Telecommunications
Research Institute

ETRI의 최첨단 기술은 새로운 영역과 새로운 가치를 만들어내고 있습니다.
꿈꾸는 모든 것을 현실로 구현하는 기술. ETRI의 첨단 ICT기술은 세상을 더욱 풍요롭게 만들어 갑니다.