코드 복잡도 해결을 위한 Python 정적 분석기 개발

홍제성*[1], 김영철*[2]
*홍익대학교 일반대학원 소프트웨어공학연구실
**홍익대학교 소프트웨어융합학과
e-mail: *hong@selab.hongik.ac.kr, **bob@hongik.ac.kr

Python's Static Analyzer for solving Code Complexity

Je Seong Hong*[1], R.Young Chul Kim*[2]
*SE Lab., Graduate school , Hongik University

요 약

앞으로 4 차 산업혁명 시대에 많은 인공지능 관련 소프트웨어 및 데이터 기반 소프트웨어가 개발이 필수적이다. 문제는 이런 소프트웨어 관련 품질을 고려하지 않고 있다. 또한 많은 Python 관련 공개 소프트웨어에 대해 품질 보장이 불가능하다. 이를 위해, 코드 가시화 메커니즘, 인공지능 관련 코드 품질을 높이기 위해 AI 관련 Python 코드 복잡도 기반 고품질화 및 코드 가시화 메커니즘을 제안한다. 또한 기존의 복잡도를 측정하는 품질 메트릭스 중 하나인 McCabe's Cyclomatic 복잡도의 개선을 제안한다. 기존의 복잡도 공식에 응집도, 결합도를 가중치로 적용하여 개선된 복잡도를 계산한다. 소프트웨어의 내부 구조 및 관계와 복잡도 정보를 가시화하여 소프트웨어의 품질 향상에 기여한다.

1. 서론

최근 인공지능 기술의 발전으로 다양한 산업과 융합하면서, 인공지능을 결합한 소프트웨어의 개발이 많아지고 있다. 이와 함께 소프트웨어의 규모와 복잡도가 증가하여 소프트웨어 유지보수가 점점 어려워져 인공지능 소프트웨어의 고품질화를 위한 방법이 필요하다. 복잡도는 소프트웨어 성능과 안정성에 악영향을 끼치기 때문에 소프트웨어 품질에 중요한 요소이다. 따라서, 복잡도를 낮추어야 한다. 또한, 소프트웨어는 비가시성의 특징을 갖는다. 복잡한 소프트웨어의 구조와 모듈 내부, 외부 간의 상호작용 형태를 쉽게 알 수 없다. 그리고, 기존의 McCabe's Cyclomatic 복잡도 지표만을 적용한 정적 분석은 한계가 있다. 이를 해결하기 위해 가중치를 적용한 통합된 지표를 적용한다.

본 논문에서 분석할 코드는 인공지능 프로그램을 개발하는데 많이 쓰이는 Python 언어를 대상으로 한다. Python 코드를 White Box 기반의 정적 분석으로 고품질 인공지능 소프트웨어를 위해 기존의 Python 정적 분석기[1]에서 다룬 여러 코드 품질 지표 메트릭스와 모듈 내부의 응집도(Cohesion), 모듈 간의 결합도(Coupling)를 활용하여 프로그램을 가시화한다.

본 논문의 구성은 다음과 같다. 2 장에서는 관련연구로 기존의 McCabe's Cyclomatic 복잡도와 결합도, 응집도를 설명한다. 3 장에서는 2 장에서 언급한 항목들을 가중치로 적용하여 통합한 복잡도를 설명한다. 4 장에서는 기존의 정적 분석기에 통합된 복잡도를 적용하여 가시화한 결과를 보인다. 마지막으로 5 장에서 결론 및 향후 연구를 기술한다.

2. 관련연구

2.1. McCabe's Cyclomatic 복잡도

McCabe's Cyclomatic 복잡도[2]는 소스 코드의 복잡도를 정량적으로 측정하기 위한 지표 중 하나이다. 아래 그림[1]은 코드의 McCabe's Cyclomatic 복잡도를 계산하는 예시를 나타낸다.

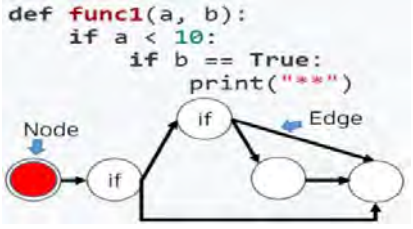

그림 1 McCabe's Cyclomatic 복잡도

복잡도 $V(G) = E - N + 2P$ 공식으로 E는 그래프의 엣지, N는 노드, P는 연결되어 있지 않은 그래프의 부분 수로 계산한다. 따라서 위 그림[1]의 코드 그래프는 6(Edge) – 5(Node) + 2 = 3 으로 계산된다.

Cyclomatic 복잡도는 높을수록 프로그램의 수정이 어려워 결함의 가능성이 높고, 비구조적인 형태로 불안정한 소프트웨어로 측정된다.

2.2. 결합도

소스 코드 구조의 결합도가 낮을수록 좋은 소프트웨어라 할 수 있다. 결합도는 소프트웨어의 모듈 요소가 다른 모듈에 얼마나 의존적인지를 의미한다. 결합도가 높은 의존적 구조의 소스 코드는 다른 모듈의 변경이 발생하면 함께 변경이 이루어져야 하기 때문에 소프트웨어의 재사용성과 유지보수에 악영향을 끼친다. 아래 표[1]은 결합도의 내용을 나타낸다.

표 1 결합도의 정의

종류	정의
자료	모듈 간의 인터페이스가 자료 요소로 전달될 때의 결합도
스탬프	모듈 간의 인터페이스로 배열이나 레코드 등의 자료 구조가 전달될 때의 결합도
제어	한 모듈에서 다른 모듈의 논리 흐름을 제어하는데 쓰이는 요소가 전달될 때의 결합도
외부	어떤 모듈에서 외부로 선언한 데이터를 다른 모듈에서 참조할 때의 결합도
공통	공유되는 공통 데이터 영역을 여러 모듈이 사용할 때의 결합도
내용	한 모듈이 다른 모듈의 내부 기능 및 내부 자료를 직접 참조, 수정할 때의 결합도

위 표[1]에서 결합도는 자료 결합도에서 내용 결합도 순으로 그 결합한 정도가 높아진다.

2.3. 응집도

응집도는 결합도와 대비되는 것으로 높은 응집도일수록 좋은 소프트웨어가 된다. 이는 모듈이 독립적으로 잘 응집되어 한 기능을 수행하는 정도를 나타낸다. 응집도가 높을수록 모듈의 재사용성이 높아진다. 아래 표[2]는 응집도의 내용이다. 응집도는 기능적 응집도에서 우연적 응집도 순으로 응집한 정도가 낮아진다.

표 2 응집도 종류 및 정의

종류	정의
기능적	하나의 모듈이 하나의 기능만 수행하는 정도
순차적	모듈 내의 동일한 요소에 값을 출력하고 다시 입력 값으로 사용하는 정도
통신적	모듈 내 요소들이 동일한 자료를 이용하여 서로 다른 기능을 수행하는 정도
절차적	모듈 내 요소 간 실행되어야 하는 순서가 존재하는 정도
시간적	변수들이 처음 초기화 되는 것 같이 모듈에서 동시에 실행되는 요소가 있는 정도
논리적	논리적으로 유사 기능을 수행하지만 서로 관련이 없는 경우
우연적	위의 경우가 모두 아닌 경우

3. 가중치를 적용하여 통합한 복잡도

소프트웨어의 복잡도 증가는 소프트웨어의 개발 과정에서의 결함 및 운영 도중에 결함을 발생시킬 가능성이 높다. 많은 개발자들이 알고 있듯이 소프트웨어 모듈의 개발은 낮은 결합도와 높은 응집도를 갖게 하는 것이 바람직하다. 그러나, 기존의 McCabe's Cyclomatic 복잡도는 코드의 메소드 내부의 분기와 같은 로직을 기반하여 분석한다. 같은 구조를 갖는 코드를 Cyclomatic 복잡도를 적용하면, 계산한 값은 같으나, 각각의 결합도를 적용하여 계산하면 다른 결과값이 나온다[3]. 따라서 모듈, 클래스 관점에서 적용하기에는 한계가 있다. 아래 그림[2]는 위 설명처럼 기존의 McCabe's Cyclomatic 복잡도를 적용했을 경우와 각각의 결합도 분석을 적용한 것의 차이를 나타낸다.

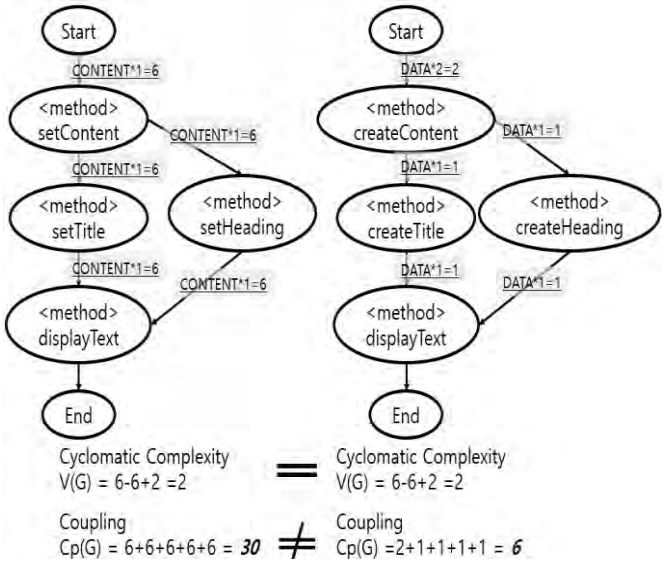

그림 2 McCabe's Cyclomatic 복잡도 분석의 한계

같은 구조의 코드를 Cyclomatic 복잡도를 적용하였을 때의 복잡도는 '2'로 같지만 각 코드에 Content 결합도, Data 결합도를 적용하면 '30' ≠ '6'으로 다른 수치로 나타나게 된다.

본 논문에서는 이를 모듈 및 클래스 관점에서 분석 및 적용하기 위해 결합도와 응집도를 통합하여 적용한다. 기존 McCabe's Cyclomatic 방식의 메소드 내부를 분석하는 것과 달리 각 노드들은 프로그램의 모듈이다. 각 노드에서는 모듈 내부의 응집도를 측정한다. 그래프의 엣지는 프로그램 모듈 사이의 결합을 나타낸다. 즉, 로직 기반의 복잡도 공식인 $V(G) = E - N + 2P$가 아닌 모듈, 클래스 관점에서의 복잡도를 분

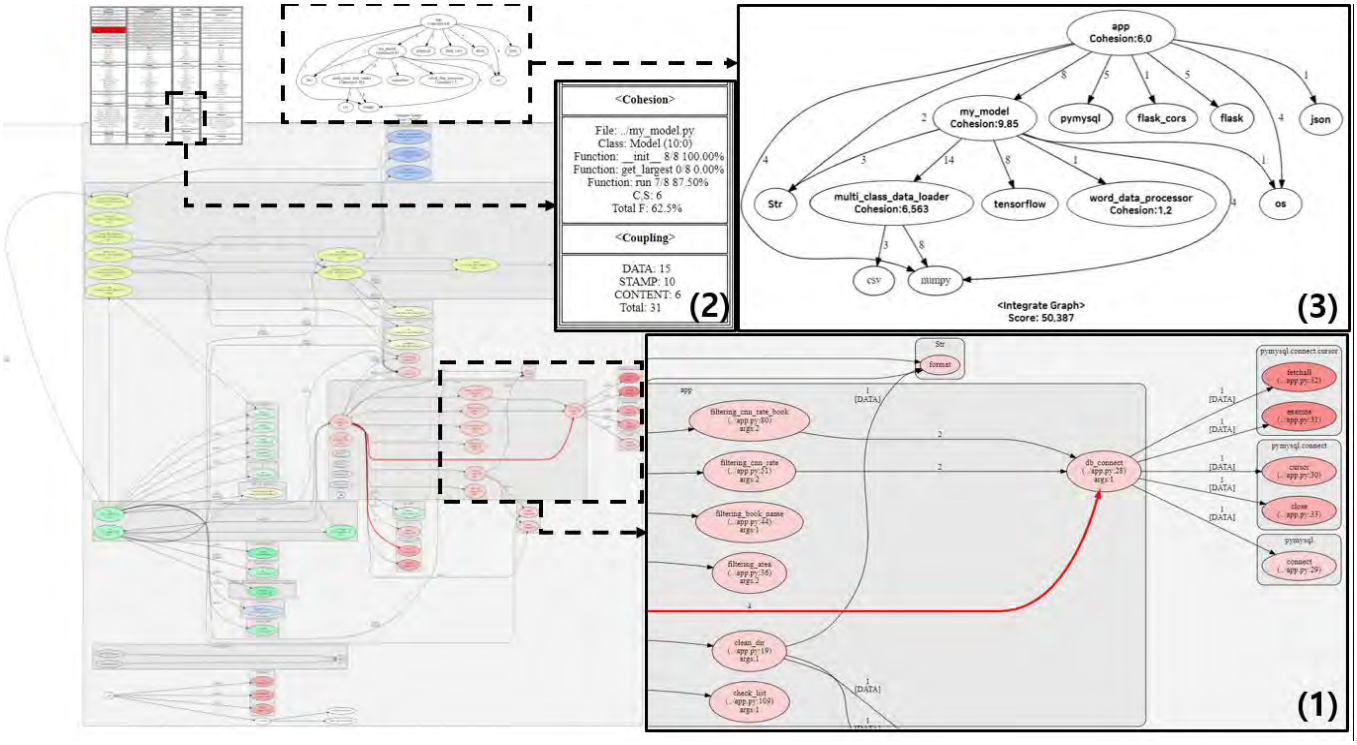

그림 3 통합된 가중치를 적용하여 개선한 Python 정적 분석기 결과 그래프

석하기 위해 $V(G) = (E * Cp) - (N * Ch) + 2$ 식과 같이 모듈 사이의 엣지는 모듈 간의 결합도를 적용한 가중치를 적용하여 $(E * Cp)$를, 각 노드가 되는 모듈에는 모듈의 응집도를 적용한 $(N * Ch)$를 적용하여 통합한 복잡도 계산을 할 수가 있다. 기존의 Cyclomatic 방식에서는 노드를 하나의 태스크로 표현하고, 엣지를 태스크 간 제어 흐름이라 한 것을 변경하여, 하나의 응집된 기능의 모듈을 노드로, 모듈 간 호출 흐름을 엣지로 보는 관점이다. 따라서, 계산된 복잡도의 수치가 낮을수록 좋은 소프트웨어라 판단할 수 있다.

본 논문에서 적용한 결합도 항목은 자료, 스탬프, 공통, 내용 결합도를 적용하였으며, 응집도 항목으로 기능적, 순차적, 통신적 응집도를 적용한다.

4. 기존의 정적 분석기에 통합된 복잡도 가시화 결과

기존의 Python 정적 분석기는 프로그램의 구조와, 모듈 내부 및 외부의 호출 관계와 횟수, 파라미터 개수와 여러 품질 메트릭스(McCabe's Cyclomatic Complexity, Raw Metrics, Halstead Metrics, Functional Cohesion)를 사용하여 가시화를 하였다. 본 논문에서 언급하는 개선한 정적 분석기에 Text-CNN 을 활용한 교육 프로그램 추천 프로그램[4]을 분석 코드로 입력하여 위 그림[3]의 결과를 나타낸다. 설명은 다음과 같다. (1) 입력한 프로그램의 내부 구조를 기존의 정적 분석기의 기능과 함께 자세히 가시화하며, 모듈 간의 호출 횟수와 결합도 정보를 그래프의 연결 실선으로 나타낸다. (2) my_model.py 의 Model 클래스에서 추출된 응집도 값 C, S, Total F(Communicational=3, Sequential=3, Functional=62.5%)와 결합도 값(Data=15, Stamp=10, Content=6)을 표 형식으로 출력되어 볼 수 있다. (3) 프로그램 전체의 결합도 및 응집도 정보를 한 눈에 보도록 가시화한다. 그래프의 노드는 프로그램 모듈로 모듈 내부의 응집도 점수를 보인다. 연결 실선은 모듈간 결합도 관계와 점수이다. 따라서, 분석 대상 프로그램의 총 복잡도는 다음과 같이 계산할 수 있다.

- 총 응집도: $6.0 + 9.85 + 6.563 + 1.2 = 23.613$
- 총 결합도: $8 + 5 + 1 + 5 + 1 + 2 + 4 + 4 + 3 + 14 + 8 + 1 + 1 + 4 + 3 + 8 = 72$
- 총 복잡도: $72 - 23.613 + 2 = 50.387$

추출되는 각각의 응집도와 복잡도에 적용되는 점수는 아래 표[3]과 같이 설정한다.

표 3 응집도와 결합도 적용 점수

종류	점수
자료(결합도)	+1
스탬프(결합도)	+2
공통(결합도)	+5
내용(결합도)	+6
기능적(응집도)	+(응집정도(%) * 0.1)
순차적(응집도)	+0.6
통신적(응집도)	+0.6

따라서, 분석기를 통해 가시화된 위 그림[3]의 결과 그래프를 토대로 호출 횟수 제약 조건을 넘어 빨갛게 표시된 부분, 개발한 모듈이 다른 모듈과 얼마나 의존적으로 개발되었는지 쉽게 인지할 수 있다. 분석 대상으로 입력된 소프트웨어의 전체 복잡도 Score 는

50 이상으로 분석되었으며, 다른 모듈과 의존적이며 응집도가 낮은 app.py 모듈과 my_model.py 모듈의 결합도를 낮추고 응집도를 높여 수정하는 작업이 필요함을 알 수 있다.

5. 결론 및 향후 연구

인공지능 소프트웨어의 발전으로 소프트웨어의 규모와 복잡도가 크게 증가하는 시점이다. 인공지능 소프트웨어의 고품질화를 위해 Python 프로그래밍 언어로 작성된 소프트웨어의 복잡도를 분석하고 가시화한다. 본 논문은 모듈, 클래스 관점에서의 복잡도 추출을 위해 기존의 McCabe's Cyclomatic 복잡도에 가중치를 적용하여 계산한다. 분석된 결과 그래프를 통해 프로그램의 설계 확인과 복잡도가 높은 모듈을 확인할 수 있어 어느 부분을 수정하고 리팩토링을 할지 판단이 가능하다. 따라서 프로그램의 유지 보수에도 도움이 될 것으로 기대한다. 향후 연구로 결합도와 응집도의 추출 종류를 늘려 더 정확한 분석을 하도록 할 예정이다.

Acknowledge

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

참고문헌

[1] 홍제성, 박보경, 장우성, 이원영, 정세준, 김영철 "Python 코드를 위한 정적 분석기 개발", KCSE 2020 논문집, 제 22 권, 제 1 호, pp.126-129, 2020. 02

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

[3] 변은영, 박보경, 장우성, 김영철, "객체 지향 패러다임에서의 코드 재사용을 위한 응집도 레벨 식별 모범 사례", 한국정보처리학회, 제 23 권, 제 2 호, pp.455-458, 2016. 11

[4] 홍제성, 박보경, 곽제일, 손현승, 김영철, "TextCNN 알고리즘 적용한 교육장터 플랫폼 기반 맞춤형 교육 컨텐츠 추천 메커니즘 개발", 한국정보처리학회, Vol. 26, No. 2, pp. 965-967, 2019.