https://doi.org/10.5392/JKCA.2022.22.10.754

파이썬 딥러닝 응용의 코드 리팩토링 특성 분석

Analyzing Characteristics of Code Refactoring for Python Deep-Learning Applications

김동관
목포해양대학교 컴퓨터공학과

Dong Kwan Kim(dongkwan@mmu.ac.kr)

요약

코드 리팩토링은 소프트웨어 시스템의 코드를 변경함으로써 새로운 요구사항 반영, 버그 수정, 코드 구조화 등을 달성하기 위한 유지보수 활동이다. 리팩토링 유형, 리팩토링 효과, 지원 도구 등에 관한 다양한 연구가 진행 중이다. 하지만, 많은 연구들이 자바 응용들을 대상으로 하고 있으며 파이썬 응용에 관한 리팩토링 연구는 사례가 많지 않다. 본 논문은 파이썬으로 개발된 딥러닝 시스템을 대상으로 단일 리팩토링과 복합 리팩토링을 식별하고 특성을 분석하였다. 또한, 딥러닝 응용과 일반 파이썬 응용 두 그룹에서 단일 및 복합 리팩토링 연산의 발생 빈도에 있어 통계학적 유의미한 차이가 있음을 확인하였다. 또한, 커밋 메시지의 키워드를 분석하여 소프트웨어 개발자들의 리팩토링 의도가 커밋 메시지에 반영되었는지를 분석하였다.

■ 중심어 : | 리팩토링 | 딥러닝 | 코드 변경 | 레파지토리 마이닝 | 파이썬 |

Abstract

Code refactoring refers to a maintenance task to change the code of a software system in order to consider new requirements, fix bugs, and restructure code. There have been various studies of refactoring subjects such as refactoring types, refactoring benefits, and CASE tools. However, Java applications rather than python ones have been benefited by refactoring-based coding practices. There are few cases of refactoring stuides on Python applications. This paper finds and analyzes single refactoring operations and composite refactoring operations for Python-based deep learning systems. In addition, we find that there is a statistically significant difference in the frequency of occurrence of single and complex refactoring operations in the two groups of deep learning applications and typical Python applications. Furthermore, we analyze keywords of commit messages to catch refactoring intentions of software developers.

■ keyword : | Refactoring | Deep Learning | Code Changes | Repository Mining | Python |

I. 서 론

소프트웨어 리팩토링(Software Refactoring)은 소프트웨어 구조를 향상 시켜 잠재적인 오류를 방지하거나 새로운 요구사항을 반영하는 코드 변경 작업을 의미한다[1]. 지속적인 통합을 강조하는 최근의 소프트웨어 개발 방법 관점에서 리팩토링 기술은 요구사항이나 기술의 변화에 유연하고 강건하게 대응하는 방법이다. 리

* 본 논문은 2021년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2021R1I1A3056172)
접수일자 : 2022년 09월 01일
수정일자 : 2022년 10월 04일
심사완료일 : 2022년 10월 04일
교신저자 : 김동관, e-mail : dongkwan@mmu.ac.kr

팩토링 기술은 다양한 프로그래밍 언어나 운영 플랫폼에 적용될 수 있다. 대중적 인지도가 높은 C/C++나 자바 언어로 개발된 응용에 리팩토링 연산을 적용한 연구들이 주류를 이루고 있다. 운용 플랫폼은 PC에서 태블릿, 스마트폰과 같은 모바일 장치, IoT 환경까지 리팩토링의 효과에 관한 연구가 진행되고 있다.

최근에 머신러닝/딥러닝 기술의 발전으로 이를 지원하는 파이썬 프로그래밍 언어가 크게 주목받고 있다. 머신러닝/딥러닝 개발 프레임워크나 알고리즘들이 파이썬으로 개발되고 개방형 레파지토리인 깃허브를 통해 전 세계 소프트웨어 개발자에게 공유된다. 하지만, 파이썬 딥러닝 응용들의 신속한 보급과 달리 해당 시스템에 대한 코드 리팩토링 기법에 관한 연구는 기존 응용들에 비해 활발하지 않은 실정이다. 본 논문은 깃허브에 공개된 파이썬 딥러닝 응용에 적용된 리팩토링 연산을 탐지하고 특성 및 경향을 분석한다. 파이썬으로 개발된 소프트웨어 시스템에 이미 적용된 리팩토링 연산의 특성과 커밋 메시지에 나타난 개발자들의 리팩토링 의도 등을 분석한다.

본 논문은 다음과 같이 구성된다. 2장에서는 관련 연구를 소개한다. 3장에서는 리팩토링 특성 탐지를 위한 연구 내용 및 방법에 관해 기술한다. 연구 질문에 대한 답변 및 분석은 4장에서 기술한다. 5장은 실험에서 발생할 수 있는 타당성 위협 요인을, 6장에서 실험에 대한 결론과 향후 연구방향을 정리한다.

II. 관련 연구

코드 리팩토링 기술에 대한 개념 정립은 M. Fowler [1]에 의해 이루어졌으며 이후 다양한 개발언어와 운영 플랫폼을 대상으로 리팩토링 탐지, 리팩토링 추천 및 예측, 유지보수 효과성, 코드 검토 관점에서 리팩토링 연구가 진행되고 있다. 일반적으로 리팩토링 연구는 기존에 개발된 응용의 소스 코드가 요구된다. 깃 기반의 개방형 코드 레파지토리는 리팩토링 연구를 위한 다양한 도메인의 충분한 분량의 코드가 제공되므로 이를 데이터 세트로 사용하는 사례가 증가하고 있다. 깃 기반의 프로젝트를 대상으로 수행된 리팩토링 유형, 리팩토링 특성 및 경향을 마이닝하는 작업이 수행된다.

개발자들이 리팩토링 연산을 수행하는 의도 중 하나는 잠재적 오류 발생 요소인 코드스멜을 제거하는 것[2-4]으로 이를 좀 더 효과적으로 수행하기 위해 자바 시스템을 위한 리팩토링 연산을 추천하는 기법이 제안되었다[5][6]. A. Brito는 리팩토링 그래프[7]를 이용하여 시간적인 관점에서 리팩토링 연산을 평가하였다. 단일 커밋보다는 다중 커밋에 걸쳐 수행되는 리팩토링 연산들에 대한 특성을 분석하였다. 또한, 복합 리팩토링을 여섯 가지로 분류하여 카탈로그를 제안하였다[8]. L. Sousa는 복합 리팩토링에 대한 정의, 개념, 분류, 관련 패턴 등에 대해 제안한다[9]. 복합 리팩토링에 대한 개념적인 프레임워크를 제시하여 형식적 표현, 분류, 특성을 기술하였다. 또한 코드스멜 제거에 있어 복합 리팩토링의 효과를 측정하였다. A. C. Bibiano[10]는 복합 리팩토링이 코드스멜에 미치는 영향을 정량화하여 평가하였다. 복합 리팩토링 정의, 분류하고 자바 응용에 적용된 코드스멜 제거 사례를 통해 복합 리팩토링의 효과를 제시하였다. 최근에는 머신러닝 기법을 적용하여 리팩토링 유형을 사전에 예측한다. P. S. Sagar는 자바 프로젝트를 대상으로 깃에 사용된 커밋 메시지와 소스 코드 메트릭스를 입력 데이터로 사용하여 리팩토링 유형을 예측하는 머신러닝 모델을 제공한다[11]. M. Aniche는 다양한 머신러닝 알고리즘을 사용하여 리팩토링 연산을 위한 모델을 작성하였으며 각 모델의 성능을 평가하고 모델의 입력이 되는 메트릭스를 제시하였다[12].

앞에서 언급한 연구들은 주로 자바 응용을 대상으로 리팩토링 연산에 대한 경향이나 특성을 분석하였다. 자바 응용에 적용한 리팩토링 분석 기법들이 파이썬 응용에도 적용될 수 있을 것이며 파이썬 응용이 리팩토링 혜택을 받을 수 있을 것이다.

H. Atwi는 파이썬 프로젝트에 수행된 리팩토링 연산을 탐지하는 도구를 개발하였다[13]. M. Dilhara는 파이썬으로 개발된 머신러닝 응용들을 대상으로 반복적으로 발생하는 코드 변경의 특성, 패턴을 분석하였다[14]. 대규모의 파이썬 머신러닝 응용에 포함된 리팩토링 연산의 경향을 분석했다는 점에서 의미가 있다. 파이썬 응용에 대한 리팩토링과 더불어 코드스멜 탐지 기

법에 대한 연구가 수행되고 있다. Z. Chen[15]은 파이썬 코드 코퍼스에 포함된 코드스멜을 탐지하기 위한 소프트웨어 메트릭스 기반의 탐지 기법을 제안하였다. 전통적인 코드 메트릭스와 파이썬 언어의 특성을 고려한 메트릭스가 소개된다. H. Jebnoun[16]는 파이썬 딥러닝 응용의 코드스멜을 탐지하고 특성을 분석하였다. 일반 파이썬 응용과 파이썬 딥러닝 응용의 두 종류의 데이터 세트를 통해 각 그룹의 코드스멜 특성 및 발생 경향을 분석하였다.

III. 연구 내용 및 방법

1. 리팩토링 연산 종류

[그림 1]은 “Extract Method” 리팩토링 연산의 예이다. 상단은 리팩토링 연산이 적용되기 이전의 코드이며 특정 메소드에 포함된 일부 코드를 나타낸다. 하단은 리팩토링 연산이 적용된 이후를 나타낸다. 이전 코드 블록이 새로운 메소드 construct_new_object()로 변경되었다. 데코레이터 “@staticmethod”가 새로운 메소드에 그대로 적용되었다.

```
69   @staticmethod
86   return torableObject(
87       key=key, data=data, description=proto.description,
     tags=tags)
```

Before Refactoring

```
138  @staticmethod
139  def construct_new_object(id, data, tags, description):
140      return StorableObject(key=id, data=data,
     description=description, tags=tags)
```

After Refactoring

그림 1. “Extract Method” 리팩토링 연산 예제

리팩토링 연산은 단일 리팩토링 연산(Single Refactoring Operation)과 복합 리팩토링 연산(Composite Refactoring Operation)으로 분류할 수 있다. 단일 리팩토링 연산은 메소드이름변경(Rename Method, RM), 매개변수추가(Add Parameter, AP), 매개변수삭제(Remove Parameter, RP), 매개변수변경(Change/Rename Parameter, CRP), 메소드추출(Extract Method, EM), 메소드인라인(Inline Method, IM), 메소드이동(Move Method, MM), 메소드풀업(Pull Up Method, PUM), 메소드푸쉬다운(Push Down Method, PDM), 반환값변경(Change Return Type, CRT), 클래스변경(Rename Class, RC)을 포함한다.

복합 리팩토링은 둘 이상의 상호 관련된 단일 리팩토링 연산으로 구성되며 클래스나 메소드에 영향을 준다. 본 논문은 L. Sousa[9]이 제안한 커밋 기반 복합 리팩토링(Commit-based Composite Refactoring)과 요소 기반 복합 리팩토링(Element-based Composite Refactoring)의 발생 빈도를 분석한다. 커밋 기반 복합 리팩토링은 2개 이상의 커밋 동안 발생하는 리팩토링 연산으로 시간상으로 관계를 맺는 단일 리팩토링 연산들로 구성된다. 단일 커밋이나 다중 커밋 동안 다수의 리팩토링 연산들이 수행될 수 있다. 요소 기반 복합 리팩토링은 복합 리팩토링 연산이 적용되는 대상에 관한 관점으로, 적용 대상이 파이썬 클래스나 메소드와 같은 특정 코드 요소에 한정된다. 물론, 클래스와 메소드 모두에 적용된 혼합형태도 발생한다. 본 논문에서는 비교적 발생 빈도가 높은 메소드 기반 복합 리팩토링을 고려한다. 메소드 기반 복합 리팩토링 연산은 Composition Extract Method (CEM), Composition Pullup Method (CPM), Decomposition Extract Method (DEM), Decomposition Inline Method (DIM), Decomposition Move Method (DMM), Decomposition Pushdown Method (DPM)로 분류할 수 있다[8]. 메소드 합성(Composition)이나 메소드 분해(Decomposition) 작업이 여러 개의 상호 연관된 단일 리팩토링 연산으로 구성된다.

본 논문은 커밋 기반 복합 리팩토링과 메소드 기반 복합 리팩토링을 고려하며 해당 리팩토링 연산들을 표현하기 위해 A. Brito가 제안한 리팩토링 그래프[7]를 사용한다. [그림 2]는 리팩토링 그래프의 예이다. 리팩토링 그래프 노드는 파일명, 클래스명, 메소드명을 표현내고 그래프 간선은 리팩토링 유형을 나타낸다.

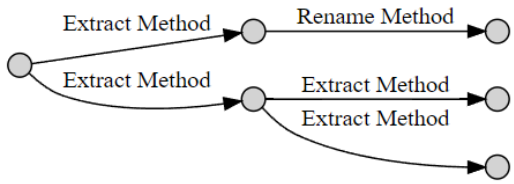

그림 2. 리팩토링 그래프의 예

[그림 3]은 대표적인 딥러닝 프레임워크 중 하나인 Keras 시스템에서 추출한 커밋 기반 복합 리팩토링을 표현한 리팩토링 그래프 예제이다. 리팩토링 그래프는 5개의 노드와 6개의 간선으로 구성되며 CS(Change Signature), EM(Extract Method) 리팩토링 연산을 포함한다. 해당 리팩토링 그래프는 세 개의 커밋(C#1, C#2, C#3) 동안 수행되었다. 두 번째 커밋(C#2)에서 4개의 Extract Method 연산이 수행됨으로써, 하나의 메소드에서 별개의 네 개의 메소드가 생성되었다.

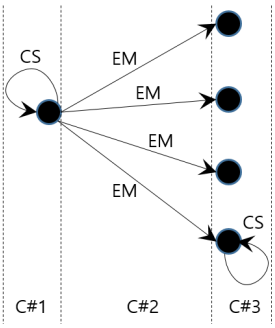

그림 3. 커밋 기반 복합 리팩토링의 예(Keras 시스템)

2. 리팩토링 연산 분석 과정 및 데이터 세트

[그림 4]는 딥러닝 시스템의 코드 리팩토링 연산에 대한 분석 과정을 나타낸다. 데이터 세트에 해당하는 파이썬 응용은 H. Jebnoun[16]에서 사용한 전체 데이터 중 일부를 사용한다. 54개의 딥러닝 시스템의 복사본을 다운로드한다. 딥러닝 시스템의 복사본을 대상으로 리팩토링 연산 및 커밋, 릴리즈와 같은 레파지토리 정보를 추출한다. 주어진 딥러닝 시스템에 적용된 리팩토링 연산들을 추출하기 위해 오픈 소스인 PyRef 도구[13]를 사용한다. PyRef는 메소드 레벨의 리팩토링 연산을 추출한다. 추출된 리팩토링 연산들은 리팩토링 그래프를 통해 방향 그래프로 표현된다. 딥러닝 시스템 전체 커밋에 대한 리팩토링 연산을 시각적으로 표현하기 때문에 여러 커밋에 걸쳐서 적용된 리팩토링 연산들을 확인할 수 있다. 딥러닝 시스템의 레파지토리 관련 정보를 식별하기 위해 PyDriller 도구[17]를 사용한다. 커밋 정보, 개발자정보, 변경 코드, 소스 코드와 같은 깃 저장소에 대한 정보를 추출할 수 있다. 커밋 정보에는 hash, 커밋 메시지, 개발자 이름, 커밋 날짜, 프로젝트 이름 등이 포함된다. 개발자들의 커밋 메시지를 통해 리팩토링 특성을 파악할 수 있다.

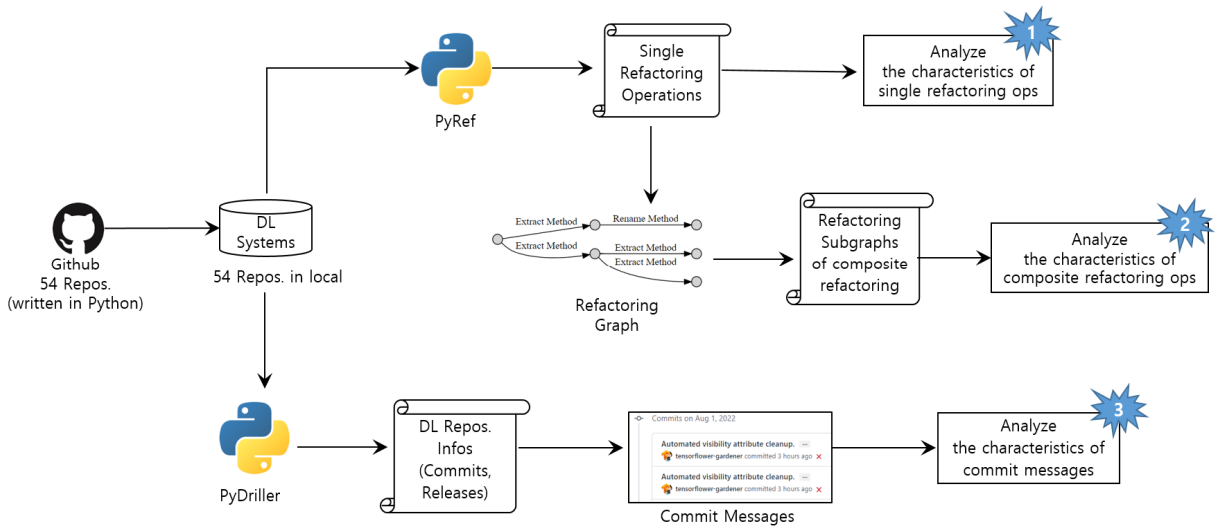

그림 4. 딥러닝 시스템을 위한 리팩토링 연산 분석 과정

데이터 세트에 해당하는 54개의 파이썬 딥러닝 프로젝트들은 [표 1]과 같이 커밋 크기에 따라 5종류의 그룹으로 분류한다. Com(G1) 〈 500는 커밋 크기가 500개보다 작은 프로젝트들은 G1 그룹에 속하다는 의미이다. "프로젝트수" 컬럼은 전체 54개 프로젝트 중 해당 그룹에 속하는 프로젝트 개수를 나타낸다. "평균커밋수"는 해당 그룹에 속하는 프로젝트들의 평균 커밋 크기를 나타낸다. 마지막 그룹인 G5에는 극단값이 포함되어 평균 커밋의 수가 다른 그룹에 비해 큰 값을 보인다. 다섯 개의 그룹에서의 리팩토링 연산의 특성을 분석한다.

표 1. 딥러닝 응용 데이터 세트 그룹화

그룹	분류기준	프로젝트수	평균커밋수
G1	Com(G1)〈500	12	292
G2	500 〈=Com(G2)〈1000	10	735
G3	1000 〈=Com(G3)〈2000	11	1,550
G4	2000 〈=Com(G4)〈3000	10	2,645
G5	3000〈=Com(G5)	11	6,384

해당 프로그램의 소스코드는 깃허브에서 호스팅되며 프로젝트 레파지토리의 대중성을 나타내는 별(Star) 수에 따라 정렬된다 (2022년 5월 기준). [표 2]는 실험을 위해 사용된 딥러닝 프로젝트에 대한 통계 정보이며 그룹별 프로젝트 별의 개수, 커밋 회수, 기여자 수, 그룹별 가장 오래된 프로젝트 날짜, 그룹별 가장 최근의 프로젝트 날짜를 표현한다. 딥러닝 응용을 위한 레파지토리는 일반 응용에 비해 비교적 최근에 생성되었음을 알 수 있다.

표 2. 딥러닝 응용 깃 레파지토리 통계 자료

그룹	Stars	Commits	Contr.	Oldest	Newest
G1	53,977	4,958	292	2015.9	2018.12
G2	39,800	7,351	509	2015.8	2018.4
G3	50,800	17,046	697	2011.9	2018.10
G4	64,605	26,453	973	2015.7	2019.1
G5	197,000	70,227	3,844	2015.3	2019.3
Total	406,182	126,035	6,315	-	-

3. 연구 질문들

본 논문은 파이썬 딥러닝 응용에 적용된 리팩토링의 특성을 분석하고자 한다. 이를 위해 깃허브에 공유된 소스 코드를 대상으로 다음과 같은 연구 질문에 답하고자 한다.

• RQ1: 파이썬 딥러닝 시스템에서 단일 리팩토링의 특성은 무엇인가?

RQ1은 파이썬 딥러닝 응용에 수행된 단일 리팩토링의 경향 및 특성과 관계한다. 단일 리팩토링은 하나의 리팩토링 연산이며 복합 리팩토링을 구성한다.

• RQ2: 파이썬 딥러닝 시스템에서 복합 리팩토링의 특성은 무엇인가?

RQ2는 커밋 기반과 메소드 기반 복합 리팩토링 연산의 특성과 관계한다.

• RQ3: 파이썬 딥러닝 시스템에서 커밋 메시지의 특성은 무엇인가?

RQ3는 리팩토링 연산이 적용된 커밋 메시지의 특성과 관계한다. 적절하게 작성된 커밋 메시지는 리팩토링 수행과 같은 개발자의 의도를 충분히 반영할 수 있다. 파이썬 딥러닝 응용과 일반 응용의 커밋 메시지를 분석한다.

IV. 연구 분석 결과

1. RQ1 결과: 단일 리팩토링 분석

파이썬 딥러닝 시스템에서의 단일 리팩토링 분포 특성을 확인하기 위해 파이썬 비딥러닝(일반) 응용 시스템과 비교하였다. 각 그룹별로 54개의 파이썬 프로젝트를 선정하여 단일 리팩토링 연산의 발생 빈도를 측정하였다. 또한, 파이썬 딥러닝 시스템에서의 유형별 단일 리팩토링 발생 빈도를 제시한다.

[그림 5]는 딥러닝 응용 그룹과 일반 응용 그룹에서의 단일 리팩토링 발생 분포를 박스 그래프로 표현한다. 먼저, 각 그룹에 속하는 파이썬 프로젝트에서 발생한 리팩토링의 수를 계산한다. 각 프로젝트 별 리팩토링 개수를 해당 프로젝트의 총 커밋 개수로 나눈다. 마지막으로 구해진 값을 0과 1사의 값으로 정규화한다. 두 그룹 간 통계학적 유의미한 차이가 있는지 검정하기

위해 비모수 Mann-Whitney Wilcoxon 검정 기법을 사용하였다. 측정결과 p-value가 2.16e-05 〈 0.05로 귀무가설이 기각된다. 즉, 단일 리팩토링 발생 빈도에 있어서, 두 파이썬 응용 집단(딥러닝 그룹, 일반 응용 그룹) 간 통계학적 유의미한 차이가 있음을 알 수 있다. 딥러닝 응용에서 일반 응용 보다 많은 단일 리팩토링이 발생함을 알 수 있다.

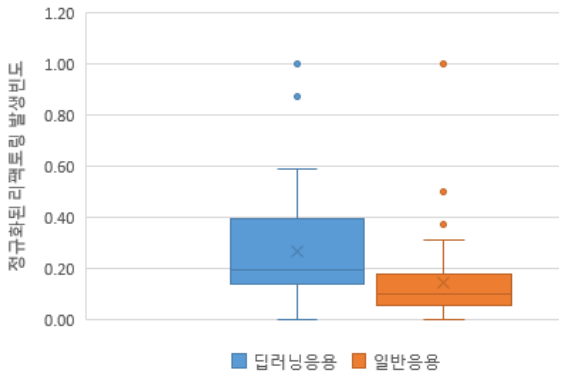

그림 5. 파이썬 응용 유형에 따른 단일 리팩토링 분포

딥러닝 응용 그룹에서 유형별 단일 리팩토링 연산 분포는 [그림 6]에 나타난다. "매개변수추가" 연산의 발생 빈도가 전체 34.3%를 차지하며 최상위에 위치한다. 그 외에 메소드이름변경, 매개변수삭제, 매개변수변경 리팩토링 연산의 발생 빈도가 높음을 알 수 있다. 매개변수추가, 메소드이름변경 리팩토링 연산이 50% 이상 발생했음을 알 수 있다. 특정 메소드의 매개변수에 대한 추가/삭제/변경 등은 자주 발생할 수 있으며 다른 리팩토링 연산에 비해 코드 변환에 의한 영향 범위가 크지 않다. 메소드의 이름 변경도 보다 의미있는 메소드 이름을 부여하거나 오타 수정으로 발생되며 상대적으로 코드 수정으로 인한 영향 범위가 크지 않다.

반면에, 메소드풀업과 메소드푸쉬다운 같은 연산은 메소드를 상위 클래스나 하위 클래스로 이동시키는 연산으로 변환 범위가 상대적으로 크다. 주어진 딥러닝 시스템에서는 0.5% 이하로 해당 연산들이 발생하였다.

[표 3]은 딥러닝 응용에 속하는 다섯 개 그룹별 (G1~G5) 단일 리팩토링 연산에 대한 분포를 나타낸다. "리팩토링 커밋수" 컬럼은 커밋 중 리팩토링 연산과 관련된 커밋들을 의미한다. 괄호 안은 전체커밋수에 대한 리팩토링커밋수의 백분율을 나타낸다. 전체 126,035개의 커밋 중 12.65%에 해당하는 15,938개가 리팩토링 연산과 관련된 커밋에 해당한다. 10.40% ~ 13.92% 구간에 리팩토링 커밋이 분포함을 알 수 있다. "리팩토링 연산수" 컬럼은 해당 프로젝트 그룹에 발생한 리팩토링 연산의 총합을 나타낸다. 일반적으로 하나의 커밋에 여러 개의 리팩토링 연산이 발생할 수 있으므로 리팩토링 커밋 총합보다 큰 값을 가진다. 커밋의 수가 증가함에 따라, 리팩토링 연산의 수도 증가함을 알 수 있다. 커밋의 수가 증가한다는 의미는 기존 코드에 대한 변경 작업이 증가함을 의미하므로, 리팩토링 연산 작업이 수행되었을 가능성이 증가하게 된다.

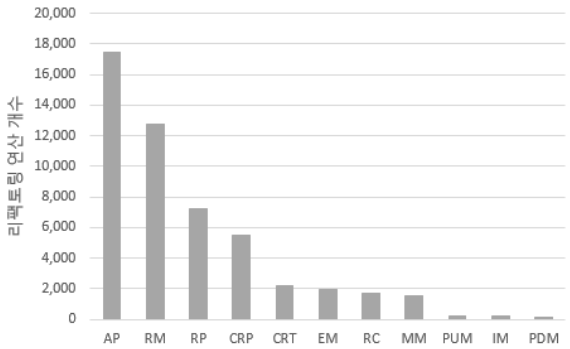

그림 6. 딥러닝 응용 그룹에 포함된 단일 리팩토링 연산 분포

프로젝트 그룹별로 11개의 단일 리팩토링 연산의 발생 분포를 알 수 있다. AP 연산이 모든 그룹에서 발생 빈도가 가장 높다. 특히, 그룹 G2의 경우, 50% 이상이 AP 연산에 해당한다. 그룹 G2를 제외하면, 최상위에 AP, RM, RP 순으로 연산들이 발생했다. 이에 반해, PUM, PDM, IM 연산은 모든 그룹에서 1%미만으로 발생했다.

2. RQ2 결과: 복합 리팩토링 분석

RQ1은 단일 리팩토링의 특성을 분석하였으며 RQ2에서는 파이썬 딥러닝 응용에 적용된 복합 리팩토링의 특성을 분석한다. 특히, 커밋 기반 복합 리팩토링과 메소드 기반 복합 리팩토링을 고려한다.

2.1 커밋 기반 복합 리팩토링

[그림 7]은 딥러닝 응용과 일반 응용에서의 커밋 기반 복합 리팩토링 발생 건수를 박스 그래프로 표현한

표 3. 딥러닝 응용에서 그룹별 단일 리팩토링 연산 분포

그룹	전체 커밋수	리팩토링 커밋수	리팩토링 연산 수	AP	RP	CRP	RM	EM	MM	IM	RC	PUM	PDM	CRT
G1	4,958	690 (13.92%)	1,949	**937 (48.08%)**	235 (12.06%)	242 (12.42%)	378 (19.39%)	43 (2.21%)	41 (2.10%)	6 (0.31%)	61 (3.13%)	4 (0.21%)	0 (0.00%)	2 (0.10%)
G2	7,351	981 (13.35%)	2,703	**1,452 (53.72%)**	514 (19.02%)	233 (8.62%)	220 (8.14%)	101 (3.74%)	47 (1.74%)	6 (0.22%)	83 (3.07%)	6 (0.22%)	8 (0.30%)	33 (1.22%)
G3	17,046	1,772 (10.40%)	4,377	**1,690 (38.61%)**	622 (14.21%)	515 (11.77%)	868 (19.83%)	216 (4.93%)	193 (4.41%)	29 (0.66%)	160 (3.66%)	9 (0.21%)	4 (0.09%)	71 (1.62%)
G4	26,453	3,518 (13.30%)	9,824	**3,419 (34.80%)**	1,489 (15.16%)	1,246 (12.68%)	1,964 (19.99%)	536 (5.46%)	254 (2.59%)	47 (0.48%)	360 (3.66%)	43 (0.44%)	63 (0.64%)	403 (4.10%)
G5	70,227	8,977 (12.78%)	28,099	**9,187 (32.70%)**	3,758 (13.37%)	2,874 (10.23%)	7,454 (26.53%)	1,069 (3.80%)	958 (3.41%)	134 (0.48%)	1,014 (3.61%)	170 (0.61%)	52 (0.19%)	1,429 (5.09%)
총합	126,035	15,938 (12.65%)	46,952	**16,685 (35.54%)**	6,618 (14.10%)	5,110 (10.88%)	10,884 (23.18%)	1,965 (4.19%)	1,493 (3.18%)	222 (0.47%)	1,678 (3.57%)	232 (0.49%)	127 (0.27%)	1,938 (4.13%)

다. 딥러닝 응용에서 일반 응용에 비해 커밋 기반 리팩토링이 더 많이 발생함을 알 수 있다. 두 그룹 간 통계학적 유의미한 차이가 있는지 검정하기 위해 Mann-Whitney Wilcoxon 검정 기법을 사용했다. 해당 검정 기법의 p-value가 0.02 〈 0.05로 귀무가설이 기각된다. 즉, 커밋 기반 리팩토링 발생 건수에 있어서, 두 파이썬 응용 집단(딥러닝 그룹, 일반 응용 그룹) 간 통계학적 유의미한 차이가 있음을 알 수 있다. 딥러닝 응용에서 일반 응용 보다 더 많은 커밋 기반 복합 리팩토링이 발생함을 알 수 있다.

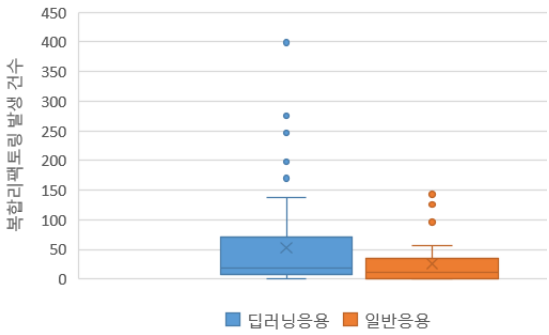

그림 7. 파이썬 응용 유형에 따른 커밋 기반 복합 리팩토링 발생 분포

파이썬 딥러닝 응용에서의 커밋 기반 복합 리팩토링 발생 분포에 대해 보다 자세하게 기술한다.

딥러닝 응용 데이터세트를 다섯 개의 그룹으로 분류하여 커밋 기반 복합 리팩토링 발생 분포를 확인한다. [표 4]는 그룹별 커밋 기반 복합 리팩토링의 분포를 나타낸다. 총 2,847개의 커밋 기반 복합 리팩토링 그래프가 생성되었음을 알 수 있다. 커밋의 개수는 2개, 3개, 4개, 5개 이상으로 분류한다. 두 개 이상의 커밋에 걸쳐서 발생한 리팩토링 연산들은 단일 시점이 아닌 시간이 지나면서 코드 변경이 발생했음을 의미한다. 그룹에 상관없이 2개의 커밋 크기를 가지는 복합 리팩토링이 대략 80% 정도 되므로, 대부분의 복합 리팩토링이 2개의 커밋 동안에 완료되었음을 알 수 있다. 그룹 G5에 속하는 리팩토링의 경우 극단값을 가지는 경우가 있어 5개 이상의 커밋 크기가 다른 그룹에 비해 발생 비율이 높음을 알 수 있다.

표 4. 커밋 개수에 따른 커밋 기반 복합 리팩토링 분포

그룹	커밋 개수				소계
	2	3	4	5+	
G1	96 (80.67%)	18 (15.13%)	4 (3.36%)	1 (0.84%)	119
G2	89 (89.00%)	9 (9.00%)	1 (1.00%)	1 (1.00%)	100
G3	226 (81.00%)	43 (15.41%)	5 (1.79%)	5 (1.79%)	279
G4	531 (79.14%)	88 (13.11%)	39 (5.81%)	13 (1.94%)	671
G5	1,294 (77.12%)	292 (17.40%)	50 (2.98%)	42 (2.50%)	1,678
총합	2,236 (78.54%)	450 (15.81%)	99 (3.48%)	62 (2.18%)	2,847

[표 5]는 커밋 기반 복합 리팩토링 그래프의 노드와 간선의 크기를 나타낸다. 2개의 노드(60.80%)와 간선(73.09%)을 가진 리팩토링 그래프가 가장 많으며, 이는 리팩토링 그래프가 비교적 단순한 경우가 많이 발생함을 의미한다.

표 5. 커밋 기반 리팩토링 그래프의 노드와 간선 분포

분류	2개	3개	4개	5개	6+개	소계
노드 수	1,731 (60.80%)	865 (30.38)	172 (6.04%)	41 (1.44%)	38 (1.33%)	2,847
간선 수	2,081 (73.09%)	505 (17.74%)	139 (4.88%)	56 (1.97%)	66 (2.32%)	2,847

[표 6]은 커밋 기반 리팩토링 그래프를 구성하는 리팩토링 연산들의 분포를 나타낸다. 총 2,847개의 커밋 기반 복합 리팩토링 그래프가 추출되었으며 7,071개의 단일 리팩토링 연산으로 구성된다. 전체 단일 리팩토링 연산 46,952개 중 15.06%에 해당한다. RM 연산(30.62%)이 그룹 G2를 제외한, 모든 그룹에서 제일 많이 발생하였다. RM 연산은 두 번째로 자주 발생하는 단일 리팩토링 연산이었다.

2.2 메소드 기반 복합 리팩토링

주어진 딥러닝 파이썬 데이터 세트에 주로 나타나는 메소드 기반 복합개념 리팩토링은 Decomposition Extract Method (DEM)와 Decomposition Move Method (DMM)이다. [그림 8]은 딥러닝 프로젝트 그룹별 Extract Method와 그 가운데 복합 리팩토링에 해당하는 DEM 발생 건수를 나타낸다. 커밋 메시지 수가 증가할수록 EM과 DEM의 발생빈도가 함께 증가함을 알 수 있다. EM 중 DEM에 해당하는 비율은 9.3%~11.38%에 분포한다. 이는 단일 리팩토링 연산 EM 중에 평균적으로 11.2% 비율로 복합 리팩토링 DEM과 관계함을 의미한다.

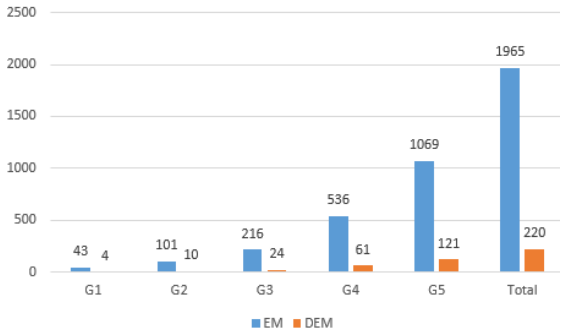

그림 8. 딥러닝 프로젝트 그룹별 DEM 발생 빈도

[그림 9]는 단일 리팩토링 연산 Move Method (MM)와 복합 리팩토링 연산 DMM의 관계를 나타낸다. MM에 따른 DMM 발생 비율은 14.63%~23.40%에 분포한다. 단일 리팩토링 MM 중에 평균적으로 18.55% 비율로 복합 리팩토링 DMM이 발생함을 의미한다. EM에서 DEM 발생 비율보다 MM에서 DMM 발생 비율이 높음을 알 수 있다.

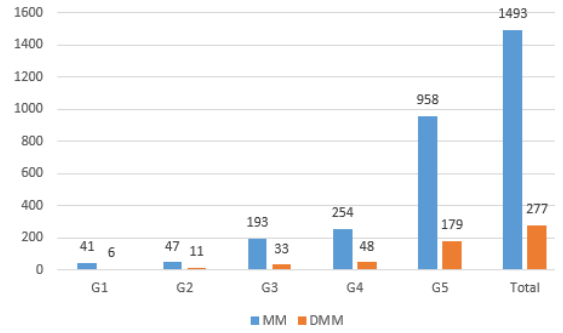

그림 9. 딥러닝 프로젝트 그룹별 DMM 발생 빈도

표 6. 커밋 기반 복합 리팩토링을 구성하는 리팩토링 연산 분포

그룹	AP	RP	CRP	RM	EM	MM	IM	RC	PUM	PDM	CRT	소계
G1	75 (26.79%)	27 (9.64%)	22 (7.86%)	**96 (34.29%)**	24 (8.57%)	16 (5.71%)	1 (0.36%)	16 (5.71%)	3 (1.07%)	0 (0.00%)	0 (0.00%)	280
G2	**63 (27.16%)**	35 (15.09%)	13 (5.60%)	55 (23.71%)	32 (13.79%)	17 (7.33%)	3 (1.29%)	8 (3.45%)	2 (0.86%)	2 (0.86%)	2 (0.86%)	232
G3	135 (19.88%)	69 (10.16%)	44 (6.48%)	**218 (31.11%)**	95 (13.99%)	77 (11.34%)	9 (1.33%)	20 (2.95%)	5 (0.74%)	1 (0.15%)	6 (0.88%)	679
G4	319 (18.93%)	163 (9.67%)	205 (12.17%)	**518 (30.74%)**	250 (14.84%)	97 (5.76%)	23 (1.36%)	37 (2.20%)	27 (1.60%)	12 (0.71%)	34 (2.02%)	1,685
G5	957 (22.81%)	415 (9.89%)	311 (7.41%)	**1,278 (30.46%)**	541 (12.90%)	286 (6.82%)	50 (1.19%)	143 (3.41%)	52 (1.24%)	26 (0.62%)	136 (3.24%)	4,195
총합	1,549 (21.91%)	709 (10.03%)	595 (8.41%)	**2,165 (30.62%)**	942 (13.32%)	493 (6.97%)	86 (1.22%)	224 (3.17%)	89 (1.26%)	41 (0.58%)	178 (2.52%)	7,071

[표 7]은 딥러닝 프로젝트 그룹별 단일 리팩토링과 복합 리팩토링 식별 결과를 정리한다. 총 126,035개의 커밋으로 구성된 데이터 세트에 발생한 단일 리팩토링 연산은 총 46,952개이다. 두 개 이상의 상호 연관된 리팩토링 연산으로 구성된 복합 리팩토링은 커밋 기반 복합 리팩토링, DEM, DMM 각각 2,847개, 220개, 277개 발생하였다.

표 7. 딥러닝 프로젝트 그룹별 실험결과 요약

그룹	단일리팩토링 연산 수	커밋 기반 CR	메소드 기반 복합리팩토링	
			DEM	DMM
G1	1,949	119	4	6
G2	2,703	100	10	11
G3	4,377	279	24	33
G4	9,824	671	61	48
G5	28,099	1,678	121	179
총합	46,952	2,847	220	277

3. RQ3 결과: 커밋 메시지 분석

커밋 메시지에 포함된 키워드 중심으로 리팩토링 연산 정보가 적절하게 표현되었는지 확인한다. 딥러닝 파이썬 응용과 일반 파이썬 응용의 커밋 메시지를 분석하였다. 커밋 메시지는 전체 커밋 메시지와 리팩토링 관련 커밋 메시지로 분류한다. [표 8]은 프로젝트별 출현 빈도가 가장 높은 상위 15개 키워드를 보여준다. 출현 빈도가 높은 키워드는 “fix”, “update”, “add”, “remove” 등이다. 일반 파이썬 프로젝트의 경우는 “merge”, “request”, “pull”와 같이 깃 저장소 명령어들이 최상위에 위치함을 알 수 있다. 프로젝트 유형이나 커밋 메시지 종류에 상관없이 대체적으로 유사한 키워드들이 커밋 메시지에 사용되었음을 알 수 있다. 본 논문에서 사용한 데이터 세트에 포함된 오픈 소스 프로젝트는 대중적으로 인지도가 있으며 참여 개발자들의 개발 경험이 초보 수준 이상으로 판단된다. 이는 참여 개발자들이 커밋 메시지 작성에 있어 일반적인 작성 지침을 따르거나 유사한 패턴으로 작성함을 알 수 있다.

커밋 메시지에 리팩토링 연산 유형에 대한 정보가 포함되었는지를 확인하기 위해 ‘refactoring’, ‘refactored’, ‘refactor’, ‘rename’, ‘add’, ‘remove’, ‘change/rename’, ‘extract’, ‘extracted’, ‘inline’, ‘move’, ‘pull’, ‘push’, ‘change’ 키워드에 대한 빈도수를 확인하였다. 해당 키워드에 출현 빈도는 매우 낮았으며 이는 커밋 메시지에 리팩토링 연산 유형에 대한 충분한 정보가 표현되지 않음을 의미한다. 검증을 위해 수작업으로 커밋 메시지들을 확인하였으며 리팩토링 연산 유형에 대한 정보는 부족했으며 개발자의 코드 변경 의도가 충분히 표현되지 않은 경우도 빈번하게 발생하였다. 전반적으로 커밋 메시지는 개발자에 의해 수행된 리팩토링 연산에 대해 충분한 정보를 포함하고 있지 않았으며 이는 커밋 메시지 자동 생성 등의 기능을 통해 보완될 수 있을 것으로 판단된다.

표 8. 커밋 메시지 유형에 따른 상위 출현 단어와 빈도

응용 유형	메시지 유형	커밋 메시지 수	상위 출현단어(빈도)
딥러닝파이썬 응용	전체 커밋 메시지	126,035	fix(43,437), update(32,750), add(31,107), merge(16,101), test(14,371), tests(11,711), added(9,927), remove(9,902), docs(9,419), request(8,799), pull(8,675), use(6,448), branch(6,126), fixed(5,937), bug(5,766)
	리팩토링 커밋 메시지	15,938	fix(10,836), add(8,163), update(6,553), tests(4,307), test(3,805), added(2,901), remove(2,646), docs(2,171), use(1,547), support(1,513), refactor(1,496), code(1,496), fixed(1,465), feat(1,438), fixes(1,299)
일반 파이썬 응용	전체 커밋 메시지	168,614	merge(35,464),request(23,386), pull(22,844), fix(20,170), add(18,916), update(15,629), branch(12,601), added(9,684), version(9,664), tests(9,348), test(9,163), use(9,031), release(8,859), changelog(8,365), remove(7,192)
	리팩토링 커밋 메시지	9,098	add(2,131), fix(1,708), tests(1,433), test(1,274), added(985), use(953), remove(913), update(879), support(745), new(600), fixes(545),code(520), make(514), fixed(423), celery(415)

V. 타당성 위협요인

본 논문의 연구 분석 결과의 타당성에 대한 위협 요인이 있을 수 있다. 주어진 파이썬 딥러닝 응용에서 리팩토링 연산들을 추출하기 위해 메소드 레벨의 리팩토링 추출도구인 PyRef 도구를 사용하였다. 도구의 추출 결과가 거짓 양성이나 거짓 음성이 있을 수 있으므로 이로 인해 실험 결과의 타당성 문제가 발생할 수 있다.

하지만, 기 보고된 PyRef의 리팩토링 추출 결과의 정밀도와 재현율이 연구 목적 달성에 있어, 수용할 수 있는 범위에 있으므로 해당 타당성 위협을 완화할 것으로 기대된다. 사용된 파이썬 딥러닝 응용의 한계점으로 인한 내적 타당성 문제가 야기될 수 있다. 해당 위협 요인을 줄이기 위해 깃 허브 저장소의 대중적 인지도, 키워드 검색, 별의 수, 커밋 수, 기여자 수 등을 고려하여 프로젝트를 선정하였다.

VI. 결론 및 향후연구방향

코드 리팩토링 기법에 관한 연구는 소스 코드 데이터세트가 필수적이며 코드 수집이 쉬운 자바나 C/C++ 언어로 개발된 시스템을 대상으로 한다. 최근에는 딥러닝 기술의 발달과 함께 딥러닝 파이썬 응용이 공유 저장소를 통해 접근이 쉬워졌다. 본 논문은 파이썬으로 개발된 딥러닝 시스템에 적용된 코드 리팩토링 연산의 특성을 분석한다. 단일 리팩토링 연산과 복합 리팩토링 연산의 경향, 커밋 메시지의 특성을 분석하였다.

단일 리팩토링 발생 빈도에 있어서, 딥러닝 응용 그룹과 일반 응용 그룹은 통계학적 유의미한 차이가 있음을 알 수 있었다. 딥러닝 응용 그룹에서 일반 응용 그룹보다 더 많은 단일 리팩토링이 발생하였다. 매개변수추가과 메소드이름변경 단일 리팩토링 연산이 50% 이상 발생하였다. 커밋 기반 복합 리팩토링 발생 건수에 있어서, 딥러닝 응용 그룹과 일반 응용 그룹에는 통계학적 유의미한 차이가 있음을 알 수 있었다. 딥러닝 응용에서 일반 응용보다 더 많은 커밋 기반 복합 리팩토링이 발생함을 알 수 있다. 총 2,847개의 커밋 기반 복합 리팩토링 그래프가 식별되었으며 대략 80% 정도가 두 개의 커밋 동안에 완료되었음을 알 수 있다. 커밋 메시지 분석에 있어서, 프로젝트 유형이나 커밋 메시지 종류에 상관없이 대체로 유사한 키워드들이 커밋 메시지에 사용되었음을 알 수 있었다. 리팩토링 연산과 관련된 키워드의 출현 빈도는 상대적으로 낮았으며 이는 커밋 메시지에 리팩토링 연산 유형에 대한 충분한 정보가 표현되지 않음을 의미한다.

본 논문은 메소드 레벨에서 나타나는 리팩토링 유형의 특성을 분석한다. 이를 클래스 레벨의 리팩토링으로 확장 적용한다면 파이썬 응용에 대한 종합적인 리팩토링 특성을 파악할 수 있을 것이다. 또한, 본 연구의 실험 결과를 반영한 리팩토링 지원 도구 개발을 향후 연구방향으로 제시한다. 추가로, 본 논문의 실험적 결과에 대해 다양한 데이터 분석 기법을 적용하여 실험 결과의 의미를 종합적으로 해석할 필요성이 제기된다.

참 고 문 헌

[1] Martin Fowler, Kent Beck, John Brant, *William Opdyke, and Don Roberts,* 1999. Refactoring: Improving The Design Of Existing Code (1st ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[2] G. Lacerda, F. Petrillo, M. Pimenta, and Y. G. Guéhéneuc, “Code smells and refactoring: A tertiary systematic review of challenges and observations,” Journal of Systems and Software Vol.167, 2020.

[3] C. Tavares, M. Bigonha, and E. Figueiredo. “Analyzing the impact of refactoring on bad smells,” In Proceedings of the 34th Brazilian Symposium on Software Engineering, 2020.

[4] C. Silva, A. Santana, E. Figueiredo, and M. A. S. Bigonha, “Revisiting the Bad Smell and Refactoring Relationship: A Systematic Literature Review,” In Proceedings of the XXIII Iberoamerican Conference on Software Engineering (CIbSE), 2020.

[5] R. Haas and B. Hummel, “Deriving extract method refactoring suggestions for long methods,” In International Conference on Software Quality, Springer, Cham, 2016.

[6] M. Shahidi, M. Ashtiani, and M. Zakeri-Nasrabadi, “An automated extract method refactoring approach to correct the long method code smell,” J. of Systems and Software, Vol.187, 2022.

[7] A. Brito, A. Hora, and M. T. Valente. “Refactoring graphs: Assessing refactoring

over time," In 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2020.

[8] A. Brito, A. Hora, and M. T. Valente, "Towards a Catalog of Composite Refactorings," arXiv:2201.04599, 2022

[9] L. Sousa, D. Cedrim, A. Garcia, W. Oizumi, A. C. Bibiano, D. Oliveira, M. Kim, and A. Oliveira, "Characterizing and identifying composite refactorings: Concepts, heuristics and patterns," In 17th International Conference on Mining Software Repositories (MSR), 2020.

[10] A. C. Bibiano, E. Fernandes, D. Oliveira, A. Garcia, M. Kalinowski, B. Fonseca, R. Oliveira, A. Oliveira, and D. Cedrim "A quantitative study on characteristics and effect of batch refactoring on code smells," In 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE, 2019.

[11] P. S. Sagar, E. A. AlOmar, M. W. Mkaouer, A. Ouni, and C. D. Newman, "Comparing commit messages and source code metrics for the prediction refactoring activities," Algorithms 14, No.10, 2021.

[12] M. Aniche, E. Maziero, R. Durelli, and V. Durelli, "The effectiveness of supervised machine learning algorithms in predicting software refactoring," IEEE Transactions on Software Engineering, Vol.48, Issue 4, 2020.

[13] H. Atwi, B. Lin, N. Tsantalis, Y. Kashiwa, Y. Kamei, N. Ubayashi, G. Bavota, and M. Lanza. "PyRef: refactoring detection in Python projects," In 2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM), IEEE, 2021.

[14] M. Dilhara, A. Ketkar, N. Sannidhi, and D. Dig, "Discovering repetitive code changes in Python ML systems," In International Conference on Software Engineering, ACM/IEEE., 2022.

[15] Z. Chen, C. Lin, M. Wanwangying, Z. Xiaoyu, Z. Yuming, and X. Baowen, "Understanding metric-based detectable smells in Python software: A comparative study," Information and Software Technology, Vol.94, 2018.

[16] H. Jebnoun, H. B. Braiek, M. M. Rahman, and F. Khomh, "The scent of deep learning code: An empirical study," Proceedings of the 17th International Conference on Mining Software Repositories, 2020.

[17] D. Spadini, M. Aniche, and A. Bacchelli. "Pydriller: Python framework for mining software repositories," In Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018.

저 자 소 개

김 동 관(Dong Kwan Kim) 정회원

- 1993년 2월 : 숭실대학교 전산학과 (공학사)
- 1998년 2월 : 숭실대학교 전산학과 (공학석사)
- 2009년 7월 : Virginia Tech 컴퓨터과학과(공학박사)
- 2013년 3월 ~ 현재 : 목포해양대학교 컴퓨터공학과 교수

〈관심분야〉 : 소프트웨어공학, 딥러닝 알고리즘, 빅 데이터 분석, 런타임 시스템