추상 구문 트리에 기반한 코드 변화 분석

이 창 공O, 나 예 원, 최 윤 호
한동대학교 전산전자공학부
{zackcglee, nayeawon, yhchoi}@handong.ac.kr
이 건 우, 최 명 석
한국과학기술정보연구원 기계학습데이터연구단
{gwlee, mschoi}@kisiti.re.kr
남 재 창
한동대학교 전산전자공학부
jcnam@handong.edu

Analysis of Code Change Based on Edit Script of Abstract Syntax Tree

Changgong LeeO, Yeawon Na, Yoonho Choi
Department of Computer Science and Electrical Engineering, Handong Global University
Gun-woo Lee, Myung-seok Choi
Department of Machine Learning Data Research, Korea Institute of Science and Technology Information
Jaechang Nam
Department of Computer Science and Electrical Engineering, Handong Global University

요 약

디버깅은 소프트웨어의 결함을 발견하고 제거하는 과정으로서 소프트웨어 유지 보수의 중요한 부분을 차지한다. 결함 발견, 재생산, 테스트 등의 세분화된 디버깅 과정 중 코드 수정 과정을 자동화하는 방법 중의 하나로 개발자가 소스 코드의 결함을 수정하는 패턴을 탐색하여 앞으로 발생할 코드 변화를 가정하는 연구들이 있다. 프로그래밍 언어는 엄격한 문법에 의존성이 높기 때문에 구문 정보를 포함하지 않은 규칙으로 만들어진 코드 변화는 정상적인 작동을 하지 않을 수 있다. 본 연구는 구문 정보를 포함한 코드 변화의 규칙을 탐색하기 위해서 Java, Python, C 언어로 작성된 260개의 오픈소스 프로젝트로부터 1,596,114개의 커밋들을 수집하여 19,822,324개의 코드 변화에서 변경 스크립트를 추출하고 요약 및 분석하여 구문 정보에 내포된 문맥적, 의미적 유사성을 활용할 수 있는 방법을 제안한다. 수집된 커밋에서 발생한 코드 변화를 군집화하여 관찰한 결과 약 90%의 코드 변화가 2개 이상의 동일한 구문 정보를 가지고 있었고, 이 코드 변화들은 의미적인 유사성을 보였다. 이를 기반으로 본 연구 방식을 이용하여 실제 결함을 수정한 코드 변화를 수집한다면 코드 수정의 디버깅 자동화 기법을 설계할 수 있을 것이라 기대한다.

1. 서 론

디버깅은 소프트웨어의 결함을 발견하고 제거하는 과정으로서 소프트웨어 유지 보수의 중요한 부분을 차지한다[1]. 과거부터 결함 발견, 재생산, 수정, 테스트 등의 세분화된 디버깅 과정을 자동화하려는 여러 연구가 있었다[2].

이 중 코드 수정(Fix) 과정을 자동화하는 방법 중의 하나로 개발자의 소스 코드 결함을 수정하는 과정에 패턴이 있음을 밝히려는 여러가지 시도가 있었다[3-4][3-11]. 이러한 패턴은 소스 코드의 결함을 수정하는 패턴을 탐색하여 앞으로 발생할 코드 변화를 가정하는 연구를 뒷받침한다[5-11]. 프로그래밍 언어는 프로그램의 작동을 위해 생성 규칙(Production Rule)에 따른 엄격한 문법(Syntax)이 존재한다. 따라서 개발 또는 유지 보수 과정에서 유사한 코드 변화가 반복적으로 발생할 수 있다. 이러한 유사한 코드 변화가 패턴으로서 정의될 수 있다면 코드 수정의 자동화에 활용될 수 있다.

과거에 소스 코드를 의미적으로 분석하기 위해 자연어 모델을 사용해 코드 변화의 비선형 패턴을 학습하여 코드 생성 및 수정에 성공한 연구가 있었다[7-11]. 그러나 자연어와 달리 프로그래밍 언어는 엄격한 문법에 의존성이 높기 때문에 구문 정보(Syntactic Information)를 포함하지 않은 규칙으로 만들어진 코드 변화는 정상적인 작동을 하지 않을 수 있다[10-11].

본 연구는 구문 정보를 포함한 코드 변화의 규칙을 탐색한다. 구문 정보를 이용하여 다양한 코드 변화를 분류하는 방법을 실험하고 각 분류 군집에 속하는 코드 수정이 문맥적, 의미적 유사도를 가지는지 분석하였다. 본 연구에서는 추상 구문 트리(Abstract Syntax Tree) 상에서 코드 변화의 구문 정보를 포함하는 변경 스크립트(Edit Script)를 생성하는 소스 코드 Differencing Tool인 GumTree를 이용하였다[12]. Java, Python, C 프로그래밍 언어로 작성된 19,822,324개의 헝크 단위 코드 변화에서 변경 스크립트를 추출하고 분석하여 구문 정보를 분류할 수 있는 방법을 탐색하였다.

코드 변화의 정보를 얻어내기 위해 260개의 오픈소스 프로젝트에 존재하는 코드 변화 기록의 구문 정보를 추상 구문 트리를 통한 변경 스크립트 이용해 추출하고 구문 정보에서 존재할 수 있는 규칙을 요약, 정리, 관찰하였다. GumTree 알고리즘으로 생성된 변경 스크립트에서 코드 변화의 구문 정보가 얼마나 구체적으로 요약되었는지에 따라 범위(Scope)를 재설정하며 한 파일에 해당하는 코드 수정을 추상 구문 트리 상의 노드 변화와 변화의 위치를 내포하는 부모 노드를 기준으로 요약한 뒤 헝크(Hunk) 단위로 연결하여 군집화(Clustering)하였다. 이때 요약된 변경 스크립트의 길이와 군집의 크기가 반비례함을 확인하였으며 길이가 길수록 해당 군집에 속하는 코드 변화가 유사할 수 있음을 확인하였다. 또한, 다양한 목적을 가진 프로젝트에서 동일한 코드 수정 구문(Syntactic Change)을 가진 코드 변화의 문맥적, 의미적 유사도를 관찰하였다. 이를 토대로 본 연구 방식을 이용하여 실제 결함을 수정한 코드 변화를 수집한다면 코드 수정의 디버깅 자동화 기법을 설계할 수 있을 것이라 기대한다.

2. 연구 방법

본 연구는 커밋 수집, 추상 구문 트리로 코드 변경 및 변경 스크립트 요약, 해시코드 부호화(Encoding) 및 군집화의 세 단계로 진행되었다. 그림 1(하단)은 본 연구의 세 단계를 도식화하여 보여준다.

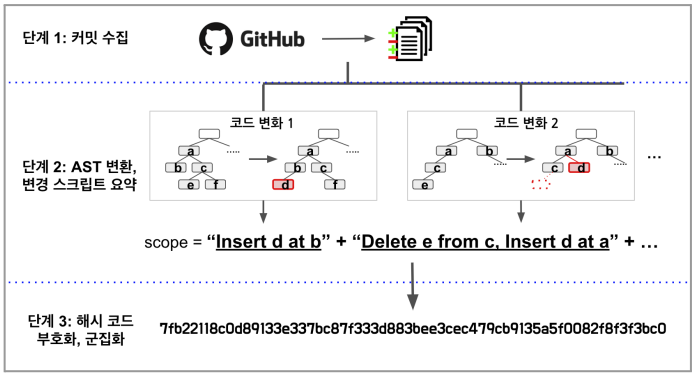

그림 **1.** 본 연구의 단계별 도식화

2.1 커밋 수집

커밋을 수집하기 위해 가장 많이 사용되는 프로그래밍 언어인 Java, Python, C 언어를 대상으로 GitHub에서 총 260개의 오픈소스 프로젝트를 선정하였다[13]. 언어별 프로젝트의 개수는 Java가 219개, Python이 25개, 그리고 C언어가 16개이다. 본 연구에서는 프로젝트의 포크 수가 많을수록 해당 소프트웨어의 관리가 활발하게 진행되었다고 가정하였다. 이 가정을 바탕으로 Java는 GitHub에 공개된 Apache 프로젝트 중에서 포크가 가장 많이 된 상위 219개의 프로젝트를 선정하였다. Python과 C 언어의 경우에는 GitHub에 공개된 오픈소스 프로젝트 중에서 포크 수가 가장 많은 순서대로 각각 상위 25개, 16개의 프로젝트를 선정하였다. 선정된 프로젝트에서 모든 커밋을 추출한 뒤 각 커밋의 변화 전과 변화 후의 소스 코드를 JGit API를 이용하여 수집하였다.

2.2 커밋 전후 코드의 AST 변환 및 변경 스크립트 요약

수집된 한 쌍의 소스 코드를 Eclipse JDT core API를 이용하여 각각 추상 구문 트리로 변환하였다. 이후 변환된 두 트리에 GumTree 알고리즘을 적용하여 두 개의 추상 구문 트리로부터 변경 스크립트를 생성하였다[12]. GumTree 알고리즘은 변수의 타입, 변수명, 코드 변경이 발생한 위치 등 프로젝트의 고유 정보를 포함하는 구체적인 변경 스크립트를 제공한다. 따라서 GumTree에서 제공하는 변경 스크립트를 그대로 사용하는 경우, 코드 변화의 구문 정보를 기준으로 군집화하는 데에 어려움이 있다. 따라서, 변경 스크립트를 보편화하기 위해 요약하는 과정이 필요하다. 이때, 요약되는 정도에 따라 변경 스크립트에서 코드의 중요한 구문 정보를 누락할 수 있기 때문에 알맞은 임계를 설정하여 코드의 구문 정보는 포함하면서 프로젝트의 구체적인 고유 정보는 제외하도록 변경 스크립트 정보를 요약해야 한다. 본 연구에서는 요약의 임계가 코드 변화가 일어난 서브 트리(Subtree)의 상위 노드(Parent Node)보다 더 상위의 정보를 포함하는 경우에 과도한 추상적 정보를 얻게 되는 것을 확인하였다. 반대로 최하위 노드를 포함하면 너무 구체적인 코드 변화 정보를 얻게 되어서 다른 코드 변경 스크립트와의 유사도를 찾기 어렵다. 본 연구에서는 변경 스크립트를 Insert, Delete, Move, Update의 수정 타입, 변화한 노드의 타입, 변화가 발생한 서브 트리의 상위 노드 정보를 포함하도록 요약하였다.

2.3 해시 코드 부호화 및 군집화

그림 1의 단계 3과 같이 동일한 파일에서 발생한 모든 노드 변화를 헝크(Hunk) 단위로 연결하여 표현하였다. 즉, 복수의 노드 변화를 소스 코드상에서의 맨 위부터 아래 방향으로 순차적으로 연결한 것이다. 그림 2(하단)는 요약된 변경 스크립트의 예시 사례이다.

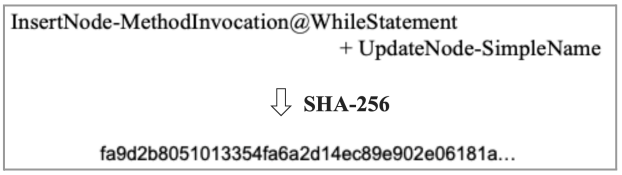

그림 **2.** 요약된 변경 스크립트와 해시 코드 변환 예시

첫 번째 헝크의 InsertNode는 수정의 타입을, Method-Invocation은 노드 타입을, WhileStatement는 수정된 서브 트리의 상위 노드 정보, 즉 변화가 일어난 위치를 뜻한다. 따라서 해당 헝크는 While문 안에 method가 호출되는 코드가 추가되었음을 의미한다. 두 번째 헝크는 update 타입이므로 수정 후 상위 노드가 변하지 않는다. 따라서 상위 노드 정보는 포함되지 않았다. 위 예시는 하나의 파일에 두 가지 헝크의 코드 수정이 있었음을 뜻한다. 이와 같은 요약된 변경 스크립트를 데이터의 복잡도를 줄이고 관리를 용이하게 하기 위해서 SHA-256 알고리즘을 이용해 해시코드로 부호화하여 그림 2와 같이 표현하였다. 부호화된 해시코드가 서로 일치하는 경우, 하나의 군집으로 묶었다. 이때 다양한 목적의 프로젝트에서 본 연구의 요약, 배열화된 변경 스크립트가 얼마나 유사한 의미를 가졌는지 비교하기 위해서 각 커밋에 해당하는 프로젝트의 이름, 커밋 아이디와 해당 코드 변화가 발생한 파일의 이름을 기본키로 사용하였다. 아래의 Algorithm 1은 본 연구 방법을 의사코드(Pseudo Code)로 표현한 것이다.

Algorithm 1 Clustering Edit Script Patterns

```
C: set of collected commits
n: number of the collected commits
C = c_1 ∪ c_2 ∪ … ∪ c_n
g_m: changes made in m^th file within the commit c_n
c_n = g_1 ∪ g_2 ∪ … ∪ g_m
P_i: i^th hash-coded edit script pattern
P_i.list: a list of repository information in P_i cluster
procedure Patternize Edit Scripts
  patterns: empty
  i ← 1
  for all c_n in C do
    changedFiles ← JGitDiff(c_n)
    for all m in changedFiles do
      AST_beforeCommits ← ASTParse(m.sourceCodeBeforeCommit)
      AST_afterCommits ← ASTParse(m.sourceCodeAfterCommit)
      g_m ← GumTreeDiff(AST_beforeCommits, AST_afterCommits)
      for all hunk in g_m
        scope += "hunk.type + hunk.changedNodes + hunk.parentNode"
      P_i.hash ← ConvertToHashCode(scope)
      if P_i.hash does not exist in patterns
        P_i.list.append(c_n.projectName, c_n.commitID, m.fileName)
        patterns.add(P_i)
        i++
      else
        patterns.P_i.list += {C_n.projectName, C_n.commitID, g_n.fileName}
  return patterns
```

표 **1.** 코드 변화 군집화 결과 통계

언어	Java	Python	C
총 프로젝트 수	219	25	16
총 커밋 수	1,219,135	327,222	49,757
총 코드 변화 수	17,589,637	2,123,880	108,807
군집 크기가 **2**개 이상인 코드 변화의 수	16,859,221 (95.8%)	2,004,352 (94.4%)	94,593 (87%)
군집 수	1,156,756	196,374	21,996
최대 군집 크기	1,031,914	112,642	4,298

평균 군집 크기	6	4	3

3. 연구 결과

앞서 제시한 방식으로 260개의 프로젝트에서 1,596,114개의 커밋들을 수집하여 총 19,822,324개의 헝크 단위의 코드 변화를 군집화하여 관찰하였다.

표 1(상단)은 실험 결과의 통계를 나타낸 것이다. 총 코드 변화 수는 프로그래밍 언어 기반 프로젝트별로 수집된 헝크 단위의 코드 변화의 개수이다. 최대 군집 크기는 가장 많은 코드 변화가 포함된 군집의 크기이다. Java의 경우 최대 1,031,914개의 코드 변화가 동일한 구문 정보를 가지고 있었다. 리팩토링(Refactoring)과 관련된 Update 타입의 단순한 코드 변화의 경우가 이에 해당된다. 해당 코드 변화는 구문 정보가 특징적이지 않아 군집이 매우 크며 전체 코드 변화의 10%에 달하는 매우 큰 크기를 가지고 있다. 평균 군집 크기는 동일한 구문 정보를 가진 코드 변화 수의 평균이다.

표 1의 결과에서 볼 수 있듯이 세 개의 언어에서 약 90%의 코드 변화들이 군집화되었다. 따라서 위의 표에 나온 대부분의 코드 변화에서 반복적으로 동일한 구문적 특징이 나타난다는 사실을 도출할 수 있다.

keras/wrappers/scikit_learn.py

```
        import copy
        import inspect
        import types
+       import numpy as np
        from ..utils.np_utils import to_categorical
        from ..models import Sequential
```

streamlit/tiny_notebook/protobuf/__init__.py

```
        from Div_pb2 import Div
        from Element_pb2 import Element
        from Delta_pb2 import Delta, DeltaList
+       from DataFrame_pb2 import DataFrame, AnyArray, Table

        # Clear out all temporary variables.
        sys.path.pop()
```

transformers/hubconf.py b/hubconf.py

```
                ...
        gpt2LMHeadModel, gpt2DoubleHeadsModel )
+       from hubconfs.transformer_xl_hubconf import (
+       transformerXLTokenizer,
+       transformerXLModel,
+       transformerXLLMHeadModel
+       )
```

그림 **3.** 군집된 코드 변화 예시 **1**

그림 3은 "InsertTree-SimpleStmt@FileInput" 군집에 해당되는 35개의 Python 코드 수정 중 3개의 예시이다. 해당 군집의 변경 스크립트는 추상 구문 트리 상에서 노드 변화가 한 번 일어났다는 의미로 SimpleStmt은 임포트와 변숫값 할당과 같은 단순한 구문을 뜻하는 노드 타입이다. FileInput은 GumTree 툴이 사용하는 parso 라이브러리에서 Python 코드를 파일로 입력받았을 때 사용하는 AST의 최상위 노드의 이름이다[14]. 즉, 위의 경우는 임포트 또는 변숫값 할당과 같은 노드의 상위 노드가 Python 코드의 최상위 노드라는 의미이다. 따라서 위의 군집은 임포트 구문 위치에 추가된 단순 구문 추가와 관련된 코드 수정 분류를 뜻한다. 세 예시 모두 모듈을 임포트하는 코드를 추가한 코드 수정이며 이와 유사한 문맥을 포함한 모듈 임포트 관련 코드 수정이 해당 군집에 14개로 전체 35개의 코드 수정 중 40% 존재했다. 나머지 60% 중 18개의 코드 수정은 변숫값 할당 관련 수정이고 나머지 2개의 수정은 소스 코드 내의 구문적 특징이 해당 군집과 무관한 수정으로 과거 연구에서 확인되었던 GumTree 알고리즘의 오류로 추측된다[15]. 과거 연구에서 기계학습 어플리케이션에서 두드러지게 발생하는 결함 패턴 중 하나로 '잘못된 임포트'가 확인된 바 있다[16]. 기계학습에서 주로 사용하는 언어인 Python에서 제공되는 다양한 기계학습 관련 라이브러리의 버전 관리가 어렵기 때문이다. 이러한 임포트 관련 코드 변화 군집의 예시는 구조가 단순하여서 코드 수정뿐만 아니라 일반적인 코드 변화에서도 유사한 코드를 추천할 때 탐색 공간을 축소할 수 있는 가능성을 보인다.

그림 4는 하나의 군집으로 묶인 Java 코드 수정 예시이다. 해당 군집에는 7개의 서로 다른 프로젝트에서 발생한 9개의 코드 변화의 구문 정보가 묶여 있으며 위의 예시는 이 중 2개의 코드 변화이다. 해당 군집의 요약된 변경 스크립트는 길이가 10이다. 다시 말해, 10개의 헝크가 연결된 형태로 추상 구문 트리 상에서 10개의 노드가 다른 위치에서 변경되었다는 의미이다. 위 예시를 포함한 모든 9개의 코드 수정에서 동일하게 하나의 메소드 로컬 변수가 필드로 선언되고, 하나의 변숫값 할당 구문이 갱신, 이동 또는 추가되었으며, 하나의 메소드가 하나 추가되었다. 해당 예시는 추상 구문 트리 상에서의 노드 변화의 개수가 실제 구문의 변화의 개수와 다를 수 있으며 이와 같은 복잡한 코드 수정에서도 문맥적 유사성이 발견되었다.

apache/axis-axis2-java-core/modules/kernel/src/org/apache/axis2/deployment/DeploymentEngine.java

```
+       protected Scheduler scheduler;
        ...
        protected void startSearch(RepositoryListener listener)
-       Scheduler scheduler = new Scheduler();
+       scheduler = new Scheduler();
        ...
+       public void cleanup() {
+           ...
+       }
```

apache/ant-ivyde/org.apache.ivyde.eclipse/src/java/org/apache/ivyde/eclipse/ui/ConfTableViewer.java

```
+       private Link select;
        public ConfTableViewer(Composite parent, int style) {
            ...
-           Link select = new Link(this, SWT.PUSH);
+           select = new Link(this, SWT.PUSH);
        ...
+       public void setEnabled(Boolean enabled) {
+           ...
+       }
```

그림 **4.** 군집된 코드 변화 예시 **2**

그림 3과 4에서 볼 수 있듯이 본 연구의 결과에서 요약된 변경 스크립트로 같은 군집에 묶인 소스 코드의 변화들이 실제로 구조적으로 유사함이 확인되었다. 또한 이러한 구조적 유사성은 문맥적인 정보를 담고 있어서 의미적으로도 유사한 정보를 담고 있을 수 있는 것이 관찰되었다. 또한 추상 구문 트리의 노드 타입이 가장 다양한 Java의 변경 스크립트가 Python의 변경 스크립트보다 분류 정확도가 높고 요약된 변경 스크립트가 복잡할수록 더욱 유사한 코드 수정이 묶일 수 있음을 관찰하였다.

추가적으로 본 연구에서 제안된 변경 스크립트 요약 방법에서 변경 스크립트의 길이가 군집 크기와 구조적 유사성을 가지는지 조사하였다. 그림 5(하단)는 Java 군집의 요약된 변경 스크립트 길이와 군집의 크기 간의 관계를 그래프로 나타낸 것이다. 과거 Nguyen et al.은 코드 변화가 일어난 추상 구문 트리의 높이(Height)를 코드 변화의 크기로 정의하여 코드 변화의 크기가 커질수록 코드 변화의 구조적 반복성이 기하급수적으로 떨어진다는 것을 실험적으로 확인하였다[3]. 이와 유사하게 표 1과 그림 5로부터 본 연구에서 제안한 변경 스크립트의 길이를 변화의 크기로 정의했을 때 변화의 구조적 반복성이 떨어지는 것을 확인하였다.

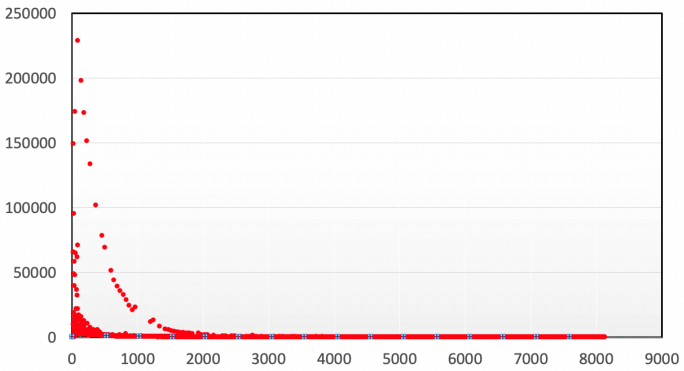

그림 **5.** 변경 스크립트의 길이(가로)와 군집크기(세로)의 관계 그래프

변경 스크립트의 길이와 같은 군집에 속하는 코드 변화의 구조적 유사성 간의 관계를 확인하기 위하여 무작위로 군집을 선정한 뒤 코드 변화를 확인하였다. 변경 스크립트의 길이가 너무 길면 소스코드가 다양한 위치에서 여러 번 수정된 된 것이다. 이때 추상 구문 트리 상의 노드 변화 순서는 매우 복잡하기 때문에 요약된 변경 스크립트가 보편화된 정보를 가지기 힘들다. 실제로 30 이상의 길이를 가진 군집의 코드 변화는 구조적 유사성이 미약함을 확인하였다. 따라서 변경 스크립트의 길이와 코드 변화의 구조적 유사성을 확인하기 위해서 길이가 1을 포함한 5의 배수인 군집 5개를 무작위로 선정한 뒤 다시 각 선정된 군집에서 최대 5개의 코드 변화를 무작위로 선정하여 코드 변화를 직접 확인하였다.

그림 6(하단)은 5의 배수만큼의 길이를 가지고 크기가 2 이상인 군집을 각각 무작위로 5개씩 선정한 뒤 각 군집에 포함된 최대 5개의 무작위로 선택된 코드 변화들의 구조적 유사도를 백분율(%)로 나타낸 것이다. 각 코드 변화의 유사성은 유사한지 여부로 이진 분류하였으며 1 저자와 2 저자의 동의 하에 문법과 문맥을 고려하여 선택된 군집 내에서 가장 지배적(Dominant)으로 발견된 구조로 비교하였다. 예를 들어 하나의 코드 변화에서 메소드의 반환 타입이 int에서 long으로 바뀌고 다른 코드 변화에서 메소드의 매개 변수 타입이 int에서 long으로 바뀌었다면 두 코드 변화는 구조적으로 유사하지 않다고 판단하였다.

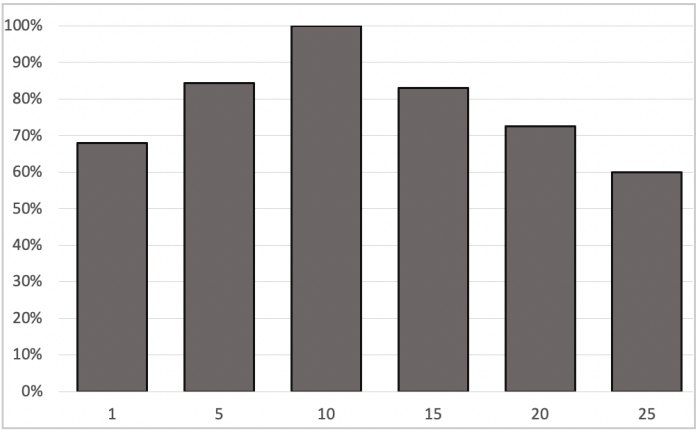

그림 **6.** 변경 스크립트의 길이에 따른 코드 변화의 유사도

구조적 유사성을 가진 코드 변화들은 의미적으로도 유사성을 가질 가능성을 보인다. 따라서 그림 6의 결과로부터 본 연구에서 제시한 변경 스크립트 요약 방법이 유효하며 구조뿐 아니라 의미적 유사도를 가질 수 있는 가능성을 보인다. 또한 코드 변화의 반복성과 구조적 유사도가 높은 변경 스크립트 길이의 범위가 존재할 수 있음을 확인하였다.

4. 기여 및 유효성

본 연구는 Java 이외에 Python과 C 언어로 작성된 41개의 프로젝트를 대상으로 수집된 376,979개의 코드 변화를 구문적으로 분석하였다. 본 연구의 결과는 기존에 조사 및 연구 대상으로 주로 사용되었던 Java뿐만 아니라 다른 언어에서도 구문적 규칙이 존재할 수 있다는 가능성을 제시한다. 본 연구에서 제시한 변경 스크립트 요약 방식은 과거의 코드 변화의 구문적 규칙성에 대한 연구들과 마찬가지로 유효하며 의미적 유사성에 대한 가능성 또한 보인다.

제안된 헝크를 차례대로 연결하여 다양한 노드 변화를 순서대로 표현하는 방식은 코드 변화의 문맥적인 정보를 내포할 수 있는 가능성을 보이지만 일부 연결된 순서가 동일한 코드 변경 패턴을 다르게 분류하게 된다. 이러한 문제를 고려하여 각 헝크에서 일어난 코드 변화를 노드 삼아 트리 형태의 데이터로 만든 뒤 최장 공통 부분 서열(Longest Common Sequence, LCS) 알고리즘으로 분류가 잘못될 수 있는 코드 변화를 탐색하는 방법을 연구 중이다.

5. 결론 및 향후 연구

본 연구에서는 추상 구문 트리와 변경 스크립트를 이용해서 코드 수정의 구문 정보를 요약했을 때 대부분의 코드 변화가 여러 가지의 반복적인 규칙을 담고 있음을 관찰하였다. 또한 제시된 방법으로 변경 스크립트를 요약하여 두 소스 코드 변화를 비교했을 때 실제의 코드가 구조적, 의미적으로 유사한 정보를 담고 있을 가능성을 제시하였다. 따라서 본 연구 방식을 이용하여 실제 결함을 수정한 코드 변화를 수집하고 변경 스크립트의 길이를 고려한다면 유사한 코드 수정들의 규칙을 발견하고 더 나아가 코드 수정의 실마리를 제공할 수 있는 가능성이 있다. 만약 이와 같은 방식을 활용하여 디버깅 자동화에 사용할 수 있다면 기존의 딥러닝을 활용한 방식이나 템플릿 기반 방식보다 비용이 낮고 성능이 뛰어난 기법을 설계할 수 있을 것으로 기대된다. 따라서 향후 연구에서는 실제 결함을 고친 코드 수정을 수집하여 새로 발생한 결함과 유사한 과거의 코드 수정을 찾을 수 있는 기법을 연구할 계획이다.

※ 본 연구는 한국과학기술정보연구원(KISTI) '기계학습 모델 개발·공유 및 코드 품질 계측 방법론 연구'의 위탁연구 과제와 정부(과학 기술정보통신부)의 재원으로 한국연구재단의 지원(No.2021R1F 1A1063049)으로 수행된 연구임

6. 참고 문헌

[1] Law, R., "An overview of debugging tools," ACM SIGSOFT Software Engineering Notes, 22, 2, pp. 43-47, 1997.

[2] Liu, K., Koyuncu, A., Bissyandé, T. F., Kim, D., Klein, J., and Le Traon, Y., "You cannot fix what you cannot find! an investigation of fault localization bias in benchmarking automated program repair systems," 12th IEEE conference on software testing, validation and verification (ICST), pp. 102-113, 2019.

[3] Nguyen, Hoan Anh, et al., "A study of repetitiveness of code changes in software evolution." *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2013.

[4] Hindle, Abram, et al., "On the naturalness of software." *Communications of the ACM* 59.5 (2016): 122-131.

[5] Kim, D., Nam, J., Song, J., and Kim, S., "Automatic patch generation learned from human-written patches," 35th International Conference on Software Engineering (ICSE), 2013.

[6] Liu, K., Koyuncu, A., Kim, D., and Bissyandé, T. F., "Avatar: Fixing semantic bugs with fix patterns of static analysis violations," IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 1-12, 2019.

[7] Zhang, J., Wang, X., Zhang, H., Sun, H., Wang, K., and Liu, X., "A novel neural source code representation based on abstract syntax tree," IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 783-794, 2019.

[8] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.D.O.,

Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G. and Ray, A., "Evaluating large language models trained on code," arXiv preprint arXiv:2107.03374, 2021.
[9] Prenner, J. A., and Robbes, R., "Automatic Program Repair with OpenAI's Codex: Evaluating QuixBugs," arXiv preprint arXiv:2111.03922, 2021.
[10] Lutellier, T., Pham, H. V., Pang, L., Li, Y., Wei, M., and Tan, L., "Coconut: combining context-aware neural translation models using ensemble for program repair," Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis, pp. 101-114, 2020.
[11] Jiang, N., Lutellier, T., and Tan, L., "CURE: Code-aware neural machine translation for automatic program repair," IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 1161-1173, 2021.
[12] Falleri, J. R., Morandat, F., Blanc, X., Martinez, M., and Monperrus, M., "Fine-grained and accurate source code differencing," Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, pp. 313-324, 2014.
[13] Delorey, D. P., Knutson, C. D., and Giraud-Carrier, C., "Programming language trends in open source development: An evaluation using data from all production phase sourceforge projects," Second International Workshop on Public Data about Software Development (WoPDaSD'07), 2007.
[14] Halter. (2017, May 8). Parso [Online]. Available: https://parso.readthedocs.io/en/latest/#resources (Retrieved 2023, Jan, 4)
[15] Matsumoto, J., Higo, Y., and Kusumoto, S., "Beyond gumtree: a hybrid approach to generate edit scripts," IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), pp. 550-554, 2019.
[16] Choi, Y, Lee, C, et al., "An Empirical Study on Defects in Open Source Artificial Intelligence Applications." *Journal of KIISE, JOK, South Korea* 49.8, 633-645, 2022.