# 커버리지 변화 정보를 활용한 코드 누락 결함에 대한 뮤테이션 기반 결함 위치 식별 기법의 성능 향상

Improving Mutation-Based Fault Localization for Better Locating Omission Faults Using Coverage Change Information

저자
(Authors)

전주영, 홍신
Juyoung Jeon, Shin Hong

출처
(Source)

정보과학회논문지 47(9), 2020.9, 863-872(10 pages)
Journal of KIISE 47(9), 2020.9, 863-872(10 pages)

발행처
(Publisher)

한국정보과학회
The Korean Institute of Information Scientists and Engineers

URL

http://www.dbpia.co.kr/journal/articleDetail?nodeId=NODE09910172

APA Style

전주영, 홍신 (2020). 커버리지 변화 정보를 활용한 코드 누락 결함에 대한 뮤테이션 기반 결함 위치 식별 기법의 성능 향상. 정보과학회논문지, 47(9), 863-872

ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 47, No. 9, pp. 863-872, 2020. 9
https://doi.org/10.5626/JOK.2020.47.9.863

# 커버리지 변화 정보를 활용한 코드 누락 결함에 대한 뮤테이션 기반 결함 위치 식별 기법의 성능 향상

# (Improving Mutation-Based Fault Localization for Better Locating Omission Faults Using Coverage Change Information)

전 주 영 [†]　　홍　신 [††]
(Juyoung Jeon)　　(Shin Hong)

**요 약** 많은 소프트웨어 결함이 코드 누락 결함(omission fault)의 형태를 가지는 데 반하여, 기존의 뮤테이션 기반 결함 위치 식별(mutation-based fault localization) 기법은 코드 누락 결함에 대해 낮은 정확도를 보여 효용성이 낮은 문제가 있다. 본 논문에서는 뮤테이션 기반 결함 위치 식별 기법인 MUSEUM과 Metallaxis에 뮤테이션에 따른 커버리지 변화와 테스트 결과 또는 출력을 연계하는 요소를 추가함으로써 코드 누락 결함을 대상으로 위치 식별 성능을 향상하는 MUSEUM+과 Metallaxis+ 기법을 제안한다. 10개의 코드 누락 결함과 6개의 일반 결함을 포함하는 총 16개 Defects4J 결함사례를 대상으로 실험한 결과, 제안한 MUSEUM+과 Metallaxis+기법은 총 10개의 코드 누락 결함 중 6개에 대해 효과적으로 정확도를 향상시켰으며, 16개 전반에 있어 정확도 향상을 확인할 수 있었다.

**키워드:** 결함 위치 추정, 뮤테이션 기반 결함 위치 식별, 코드 누락 결함, 뮤테이션 분석

***Abstract*** Although omission faults are bugs commonly found in real-world programs, existing mutation-based fault localization techniques show low accuracy at locating omission faults because useful mutants are not likely generated at locations where necessary statements are missed. This paper introduces two new techniques, MUSEUM+ and Metallaxis+, an extension of two mutation-based fault localization techniques, MUSEUM and Metallaxis, by adding elements that link the change of coverage information and the change of test results. The proposed MBFL techniques additionally utilize coverage change information to consider the characteristics of omission faults. The experiment with the 16 JFreeChart faults in Defects4J, including 10 omission faults and 6 non-omission faults demonstrate that the presented techniques, MUSEUM+ and Metallaxis+, show improved faults localization accuracy.

**Keywords:** fault localization, mutation-based fault localization, omission fault, mutation analysis

· 본 연구는 한국연구재단의 지원(NRF-2020R1C1C1013512, NRF-2017M3C4A7068179, NRF-2017R1C1B1008159)을 받아 수행했음
· 이 논문은 2019 한국소프트웨어종합학술대회에서 '코드 누락 결함(Omission Fault)에 대한 뮤테이션 기반 결함 위치 식별 기법의 성능 향상'의 제목으로 발표된 논문을 확장한 것임

† 비 회 원 : 한동대학교 일반대학원 정보통신공학과 학생
21931009@handong.edu
†† 정 회 원 : 한동대학교 전산전자공학부 교수(Handong Global Univ.)
hongshin@handong.edu
(Corresponding author임)

논문접수 : 2020년 4월 23일
(Received 23 April 2020)
논문수정 : 2020년 7월 21일
(Revised 21 July 2020)
심사완료 : 2020년 7월 22일
(Accepted 22 July 2020)


정보과학회논문지 제47권 제9호(2020. 9)

## 1. 서 론

코드 누락 결함(omission fault)이란 프로그램의 올바른 동작을 위해 특정한 위치에 존재해야 구문이 누락되어 오류가 방생하는 결함을 뜻한다. 실제 소프트웨어 개발에서 코드 누락 결함은 빈번히 발생하는 문제다[1]. 결함이 있는 구문이나 식을 삭제하거나 변경함으로써 결함수정(patch)이 이루어지는 결함과 달리, 코드 누락 결함이 있는 프로그램에 새로운 구문을 추가함으로써 결함 수정이 이루어진다는 특징이 있다.

결함 위치 추정(fault localization: FL) 기법은 개발자가 오류가 발생하는 프로그램의 결함 위치를 신속히 탐지할 수 있도록, 오류가 발생하는 프로그램을 구성하는 각 코드 요소에 결함으로 추정되는 정도인 결함 의심도(suspiciousness score)를 부여한다[2]. 뮤테이션 기반 결함 위치 추정기법(mutation-based fault localization: MBFL)은 오류 프로그램의 각 코드 요소에 변형을 가하여 변이 프로그램(mutant)을 생성한 후, 원본 프로그램과 변이 프로그램의 테스트 결과 차이 정보를 결함 의심도 추정에 사용한다[3,4]. 예를 들어, MUSEUM[3]의 결함 의심도 계산식은 특정 코드 요소에서 생성된 변이 프로그램에 의해 원본 프로그램에 대해 실패(fail)하던 테스트 케이스가 변이 프로그램에서는 성공(pass)으로 전환되는 경우가 많은 동시에 원본 프로그램에서 성공하는 테스트 케이스가 변이 프로그램에서는 실패하는 경우가 적을수록 결함 의심도를 높게 부여한다.

뮤테이션 기반 결함 위치 추정 기법은 규모가 크고 복잡한 프로그램에 대해서 높은 정확도로 결함 위치를 추정할 수 있음이 보고되고 있다[3]. 하지만, 기존의 뮤테이션 기반 결함 위치 추정 기법은 코드 누락 결함에 대해서는, 일반적인 결함에 비하여, 결함 위치 추정에 정확도가 낮다는 문제점을 가진다. 일반적인 결함과 달리, 코드 누락 결함은 결함 위치에 오류에 직접적으로 영향을 주는 구문이 존재하여서 이에 대한 변형이 오류 증상의 변형으로 이어지는 경우가 적다. 따라서, 결함 위치의 구문으로부터 결함 위치 추정에 유용한 정보를 제공하는 뮤테이션이 생성될 가능성이 적다. Defects4J[5,6]의 JFreeChart[7] 결함 집합에 존재하는 10개의 코드 누락 결함에 대하여, 실제 결함 위치를 MUSEUM의 경우 평균 203.6등, Metallaxis의 경우 평균 162.9등으로 탐지하며 비교적 낮은 성능을 보인다.

본 논문은 코드 누락 결함에 대한 뮤테이션 기반 결함 위치의 성능 향상을 위하여, 뮤테이션 발생에 따른 커버리지 변동 정보를 활용하는 새로운 MBFL 기법을 제안한다. 제안하는 기법은 기존 MBFL 기법(MUSEUM, Metallaxis)을 기본으로 하되, 원본 프로그램에서 실패하는 테스트 케이스가 변이 프로그램에서 성공으로 테스트 결과가 변경되었을 때, 원본 프로그램에서는 실행되었으나 변이 프로그램에서는 실행되지 않게 되는 구문을 식별한다. 그리고 해당 구문의 결함 의심도를 상향 조정하도록 하였다. 이러한 고려를 통해, 변이에 따른 테스트 결과 변동의 원인을 변이가 생성된 구문뿐만 아니라 실행 경로 변동에 영향을 주는 다른 구문도 함께 고려하도록 하여, 직접적인 변이 지점과의 연관성이 적은 코드 누락 결함에 대해서도 보다 정확하게 대응 가능하도록 개선하였다.

본 연구에서는 제안한 기법을 10개의 코드 누락 결함을 포함하는 총 16개 Defects4J 결함사례를 대상으로 실험한 결과, 제안한 MUSEUM+ 기법과 Metallaxis+ 기법은 각각 6개의 코드 누락 결함에 대한 정확도를 향상시켰을 뿐 아니라, 실험 대상 16개 결함 전반에 있어 정확도 성능 향상을 확인할 수 있었다. 10개의 코드 누락 결함에 대하여 MUSEUM+는 실제 결함 위치를 평균 74등으로, Metallaxis+는 38.5등으로 탐지하며 실제 결함 위치의 평균 등수에 있어서도 기존 MBFL 기법보다 높은 정확도를 갖는 것을 확인할 수 있었다.

뮤테이션에 따른 실행 동작 변화를 소프트웨어 자동 분석 연구에 활용하는 기법들이 최근 제시되고 있다. DeMiner[8]와 T-Fuzz[9]는 변이 프로그램에서 추가로 도달하는 실행 경로에 대한 정보를 추출하여 해당 경로를 원본 프로그램에서도 탐색하도록 하는 심볼릭 실행 전략을 활용하여 테스트 케이스 생성 기법의 효용성을 향상시켰다. 최근 발표된 Holmes & Groce[10]에서는 컴파일러 결함 위치 추정 방법으로 뮤테이션으로 인한 테스트 실행 결과 변화와 함께 커버리지 변화가 있는 구문을 결함 위치로 추정하는 기법을 제시하고 있다. 본 논문에서 제안하는 MUSEUM+와 Metallaxis+는 기존 MBFL에서 활용하는 뮤테이션 관련 데이터 요소와 뮤테이션으로 인한 테스트 결과 변화에 동반한 커버리지 변화 정보를 통합적으로 활용하여 결함 위치를 추정하는 기법을 제시하고, 이를 코드 누락 결함을 대상으로 분석하고 적용하였다는 면에서 기존의 연구와 차별함이 있다.

## 2. 코드 누락 결함에 대한 기존 MBFL의 한계

### 2.1 뮤테이션 기반 결함 위치 추정 기법

본 장에서는 MBFL 기법 중 MUSEUM[3]와 Metallaxis[4]으로 MBFL의 한계를 설명하고자 한다. 먼저, MUSEUM은 뮤테이션 기반 결함 위치 추정 기법으로 프로그램의 여러 변이를 만들고 변이에 테스트 케이스를 실행시켜 테스트 결과(성공, 실패)가 변이로 말미암아 변화되는 정보를 활용해 결함 위치 추정을 수행한다.

$$susp_{MSM}(s) = \frac{1}{|M(s)|}\Sigma_{m \in M(s)}(\frac{|f(s) \cap p_m|}{f2p} - \frac{|p(s) \cap f_m|}{p2f}) \quad (1)$$

MUSEUM은 식 (1)을 통해 테스트 대상 프로그램 각 구문의 결함 의심도를 계산한다. $f2p$는 모든 변이를 아울러 원본에서 실패가 변이에서 성공으로 변하는 테스트 케이스 경우의 총 수를 뜻하며, $p2f$는 원본에서 성공하던 테스트 케이스가 변이에서 실패하는 경우의 총 개수를 뜻한다. $f(s)$는 원본에서 구문 $s$를 실행시키고 실패한 테스트 케이스의 집합, $p(s)$는 원본에서 구문 $s$를 실행시키고 성공한 테스트 케이스 집합이다. $M(s)$는 구문 $s$에서 생성된 변이의 집합이며 $p_m$는 변이 $m$에서 성공하는 테스트 케이스 집합, $f_m$은 실패한 테스트 케이스 집합을 나타낸다.

식 (1)에 따라 MUSEUM은 구문 $s$의 결함 의심도 $susp_{MSM}(s)$를 (1) 원본 프로그램에서 실패하는 테스트 케이스가 구문 $s$에 변이를 한 후 성공하는 경우가 많을수록, (2) 원본 프로그램에서 성공하는 테스트 케이스가 구문 $s$에 변이를 한 후 실패하게 되는 경우가 적을수록 높게 부여한다.

Metallaxis 또한 MUSEUM과 같이 생성된 여러 변이에 테스트 케이스를 실행시키는데, Metallaxis는 변이로 인한 테스트 케이스의 결과(성공, 실패)의 변화 정보가 아닌 테스트 케이스의 출력 값(output)의 변화를 사용하여 구문의 의심도를 계산한다. Metallaxis에서는 원본 프로그램에서의 출력 값과 변이 프로그램에서의 출력 값이 다른 경우, 변이가 테스트 케이스에 의해서 killed 되었다고 정의한다.

$$susp_{Met}(s) = \max(\frac{|f(s) \cap k_m|}{\sqrt{|f| \times (|f(s) \cap k_m| + |p(s) \cap k_m|)}})_{m \in M(s)} \quad (2)$$

Metallaxis는 프로그램의 각 구문의 의심도를 식 (2)을 통하여 계산한다. 식 (2)에서는 $f$는 전체 테스트 케이스 중 원본에서 실패하는 테스트 케이스 집합을 뜻하며, $k_m$는 변이 $m$에서 원본 테스트 출력 값과 다른 출력 값을 갖는 테스트 케이스의 집합, 즉 변이 $m$을 kill하는 테스트 케이스 집합을 뜻한다. 따라서, $|f(s) \cap k_m|$와 $|p(s) \cap k_m|$는 각각 원본 프로그램에서 실패하는 테스트 케이스 중 변이 $m$을 kill하는 테스트 케이스의 개수, 원본 프로그램에서 성공하는 테스트 케이스 중 변이 $m$을 kill하는 테스트 개수를 뜻한다.

Metallaxis의 경우 구문 $s$의 결함 의심도 $susp_{Met}(s)$는 원본 프로그램에서 실패하던 테스트 케이스의 출력과 구문 $s$에 생성된 변이 프로그램에서 동일한 테스트의 출력이 다른 경우가 많을수록 높은 의심도를 부여한다.

식 (1)과 식 (2)에서 확인할 수 있는 바와 같이, MUSEUM과 Metallaxis 모두 구문 $s$의 결함 의심를 구할 때 $s$로부터 생성된 변이의 정보만을 활용하고, 다른 구문에서 생성된 변이의 테스트 결과 변화는 구문 $s$의 결함 의심도에 전혀 영향을 끼치지 않는다.

코드 누락 결함의 경우, 결함 코드가 원본 프로그램에 존재하지 않으므로 결함 코드 위치에서 결함에 직접적으로 연관된 변이를 발생시킬 가능성이 적다. 특히, MBFL에서 사용하는 변이 연산자는 원본 프로그램 코드에 존재하는 구문(statement)이나 식(expression)을 변형하거나 삭제하는 방향으로 변이를 발생시키며, 새로운 구문을 추가하는 변이 연산자는 존재하지 않는다. 변이로 인해 테스트 결과 변화의 유용한 정보가 발생하였더라도, MUSEUM과 Metallaxis는 변이 발생으로 인한 실행경로(제어흐름) 변화를 고려하지 않으므로, 코드 누락 결함을 추정하는데 한계를 가진다.

### 2.2 예제(1) - 코드 누락 결함에 대한 부정확한 결과

다음 예제는 MUSEUM이 코드 누락 결함에 대해 가지는 한계점을 구체적으로 예시한다. 아래 getDivisor는 정수를 입력받아, 2, 3, 6 중 입력 값의 가장 큰 약수 혹은 모두 서로소인 경우, 입력 값을 반환하여야 한다. 하지만, 4행과 6행 사이 코드누락으로 인해 오류가 발생할 수 있는 프로그램이다:

```
1: int getDivisor (int x) {
2:     int div=x;
3:     if (x % 2==0) {
4:         div = 2;
5:         /* if(x % 3==0) is missing */
6:         div = 6; }
7:     else if (x % 3 == 0) {
8:         div = 3 ; }
9:     return div ; }
```

위의 예제에 대한 5 개의 테스트 케이스와 각 케이스가 갖는 커버리지 정보는 그림 1에서 보인다.

| Target Program Stmt. / TC | TC1 (x=3) | TC2 (x=5) | TC3 (x=9) | TC4 (x=12) | TC5 (x=2) |
|---|---|---|---|---|---|
| 2:int div=x; | ● | ● | ● | ● | ● |
| 3:if(x % 2==0){ | ● | ● | ● | ● | ● |
| 4: div=2; | | | | ● | ● |
| 6: div=6; | | | | ● | ● |
| 7:} else if(x%3==0){ | ● | ● | ● | | |
| 8: div= 3; } | ● | | ● | | |
| 9:return div; } | ● | ● | ● | ● | ● |
| Test results | Pass | Pass | Pass | Pass | Fail |

그림 1 예제 프로그램의 테스트 케이스 실행에 따른 커버리지
Fig. 1 The test case coverage on the example program

그림 1의 첫 행에는 각 테스트 케이스의 입력 값이 주어지고 그 밑에는 해당 테스트 케이스가 각 구문을 실행시켰을 때를 검은색 동그라미(●)로 표시하였다. 마지막 행에는 해당 테스트 케이스가 성공했는지 실패했는지에 대한 정보가 있다(네 개 성공, 한 개 실패).

그림 2는 MUSEUM이 어떻게 각 구문의 결함 의심도를 계산하는지 설명한다. 그림 2의 첫 열에는 실패한 테스트 케이스가 실행한 각 구문들이 나열되어 있고 두 번째 열은 생성된 변이에 대한 정보가 있으며 세 번째 열부터 일곱 번째 열까지는 변이 프로그램에 대한 각 테스트 케이스의 실행결과 변화가 기록되어 있다. 원본에서 성공하던 테스트 케이스가 변이에서 실패로 바뀌는 경우는 'P→F', 실패에서 성공으로 변화한 경우는 'F→P'로 표시했고, 테스트 실행 결과에 변화가 없는 경우는 표시하지 않았다. 마지막 두 열은 MUSEUM이 최종적으로 계산한 각 구문의 결함 의심도와 랭킹이다.

이 예제에서 결함은 5번째 구문의 누락으로 정확한 결함 위치는 4행 혹은 6행으로 볼 수 있으나, MUSEUM은 3행에 가장 높은 결함 의심도를 주어 부정확한 결과를 냈다. 이는 3행에서 생성된 두 변이 M3와 M5로 인해 원본 프로그램에서 실패하는 테스트 케이스가 변이로 인해 성공 실행으로 변형되는 경우가 발견되었기 때문이다.

하지만, 이 예제를 살펴보면, 3번째 구문에서 생성된 두 개의 변이 M3와 M5 모두 실패 테스트 케이스를 성공하도록 만드는데, 그 이유가 3번째 구문의 변형 자체에 있다기 보다, 3행의 조건문의 변형으로 인해 4번째 구문과 6번째 구문이 실행되지 않게 되었던 것이 주요한 원인임을 알 수 있다. 이 예제는 테스트 케이스 결과를 변경하는 변이가 발생된 구문 이외에도, 변이로 인해 원본 프로그램에서는 실행되었으나 변이 프로그램 실행 시에는 실행되지 않게 된 구문도 결함 위치 분석에 중요한 정보가 됨을 보여준다. 특히, 결함을 일으키는 구문 자체에서 변이 생성이 불가능한 코드 누락 결함에 대해서는 이와 같은 테스트 실행 경로 변경 정보가 결함 위치 추정에 결정적인 역할을 할 수 있다.

### 2.3 예제(2) – 코드 누락 결함에 대한 부정확한 결과

위의 예제를 Metallaxis에 적용했을 때의 결과는 그림 3에서 확인할 수 있다. Metallaxis는 변이로 인한 테스트 케이스의 성공 또는 실패의 변화가 아닌 출력의 변화로 구문의 의심도를 측정하기 때문에 원본 프로그램의 출력 값과 변이의 출력 값을 확인해야 한다. 그림 3의 마지막 행에는 원본 프로그램에 대한 테스트 케이스의 성공 또는 실패 여부가 기록되어 있으며, 마지막에서 두 번째 행에는 원본 프로그램에 대한 테스트 케이스의 출력 값 정보가 있다. 각 구문의 변이에 대해서 세 번째 열부터 일곱 번째 열까지는 변이 프로그램의 출력 값의 변화가 기록되어 있다. 변이 프로그램의 테스트 케이스 출력 값이 원본의 테스트 케이스 출력 값과 다를 때 'killed'로 표시했다. 그림의 여덟째 열과 아홉째 열은 각각 구문을 실행시키고 결과가 성공인 테스트 케이스 중에 변이를 kill한 테스트 케이스의 개수, 구문을 실행시키고 결과가 실패인 테스트 중에 변이를 kill 한 테스트 케이스 개수를 나타낸다. 마지막 두 열은 Metallaxis로 계산한 각 구문의 결함 의심도와 랭킹이다.

Metallaxis는 결함 위치인 6행에 결함 의심도를 가장 높게 부여했지만 3행 또한 가장 높은 결함 의심도를 갖기 때문에 정확한 결과를 갖진 않는다. 각 구문에서 가장 높은 의심도를 갖는 변이는 각각 M3와 M8(또는 M9)이다. Museum과 마찬가지로 원본에서 실패하던 테

| | Mutants | TC1 (x=3) | TC2 (x=5) | TC3 (x=9) | TC4 (x=12) | TC5 (x=2) | $\lvert p(s)\cap f_m\rvert$ | $\lvert f(s)\cap p_m\rvert$ | Susp | Rank |
|---|---|---|---|---|---|---|---|---|---|---|
| 2:int div=x; | M1: x→x+1 | | **P→F** | | | | 1 | 0 | -0.04 | 3.5 |
| | M2: x→x-1 | | **P→F** | | | | 1 | 0 | | |
| 3:if(x%2==0){ | M3: ==→!= | **P→F** | **P→F** | **P→F** | **P→F** | **F→P** | 4 | 1 | 0.23 | 1 |
| | M4: ==→true | **P→F** | **P→F** | **P→F** | | | 3 | 0 | | |
| | M5: ==→ false | | | | **P→F** | **F→P** | 1 | 1 | | |
| 4: div = 2; | M6: 2→2+1 | | | | | | 0 | 0 | 0 | 2 |
| | M7: 2→2-1 | | | | | | 0 | 0 | | |
| 6: div = 6;} | M8: 6→6+1 | | | | **P→F** | | 1 | 0 | -0.04 | 3.5 |
| | M9: 6→6-1 | | | | **P→F** | | 1 | 0 | | |
| 9:return div;} | M10: div→0 | **P→F** | **P→F** | **P→F** | **P→F** | | 4 | 0 | -0.16 | 5 |
| | M11: div→-1 | **P→F** | **P→F** | **P→F** | **P→F** | | 4 | 0 | | |

그림 2 예제 프로그램에서 생성된 변이 프로그램 정보와 각 실행결과 변화 그리고 각 구문에 대한 MUSEUM의 결함 의심도

Fig. 2 The mutant information generated on the example program, change of execution result, and computed MUSEUM suspiousness score

| | Mutants | TC1 (x=3) | TC2 (x=5) | TC3 (x=9) | TC4 (x=12) | TC5 (x=2) | $\|p(s)\cap f_m\|$ | $\|f(s)\cap p_m\|$ | Susp | Rank |
|---|---|---|---|---|---|---|---|---|---|---|
| 2:int div=x; | M1: x→x+1 | | **killed** | | | | 1 | 0 | 0 | 4.5 |
| | M2: x→x−1 | | **killed** | | | | 1 | 0 | | |
| 3:if(x%2==0){ | M3: ==→!= | **killed** | **killed** | **killed** | **killed** | **killed** | 4 | 1 | 0.7 | 1.5 |
| | M4: ==→ true | **killed** | **killed** | **killed** | | | 3 | 0 | | |
| | M5: ==→ false | | | | **killed** | **killed** | 1 | 1 | | |
| 4: div = 2; | M6: 2→2+1 | | | | | | 0 | 0 | 0 | 4.5 |
| | M7: 2→2−1 | | | | | | 0 | 0 | | |
| 6: div = 6;} | M8: 6→6+1 | | | | **killed** | **killed** | 1 | 1 | 0.7 | 1.5 |
| | M9: 6→6−1 | | | | **killed** | **killed** | 1 | 1 | | |
| 9:return div;} | M10:div→0 | **killed** | **killed** | **killed** | **killed** | **killed** | 4 | 1 | 0.44 | 3 |
| | M11:div→−1 | **killed** | **killed** | **killed** | **killed** | **killed** | 4 | 1 | | |
| Test Output | | **3** | **5** | **3** | **6** | **6** | | | | |
| Test Result | | **Pass** | **Pass** | **Pass** | **Pass** | **Fail** | | | | |

그림 3 예제 프로그램에서 생성된 변이 프로그램 정보와 각 실행결과 변화 그리고 각 구문에 대한 Metallaxis의 결함 의심도

Fig 3. The mutant information generated on the example program, change of execution result, and computed Metallaxis suspiciousness score

스트 케이스가 변이 프로그램 M5에서 실제 결함 위치인 구문 4와 구문 6을 실행시키지 않았기 때문에 출력값이 변한 것이다. 그러므로, 프로그램의 실행 경로 변화가 Museum 뿐만 아닌, Metallaxis에서도 실제 결함 위치를 추적하는데 중요한 정보를 주는 것을 확인할 수 있다.

## 3. 코드 누락 결함의 특성을 고려한 새로운 뮤테이션 기반 결함 위치 추정 기법

### 3.1 변이로 인한 실행경로 변화를 고려한 MBFL

본 연구는 MUSEUM과 Metallaxis를 기반으로, 코드 누락 결함에서의 성능을 향상시키기 위해 실패에서 성공으로 테스트 결과가 변경된 테스트 실행에서 커버리지 정보를 추가로 활용한 새로운 MBFL 기법인 MUSEUM+과 Metallaxis+를 정의하였다.

$$susp_{MSM+}(s) = susp_{MSM}(s) + \frac{1}{f2p} \times \frac{\Sigma_{m \in M} |f(s) \cap \overline{p_m(s)}|}{cov_f 2nonc ov_p} \quad (3)$$

식 (3)은 MUSEUM의 결함 의심도 계산식 식 (1)에 하나의 항을 추가한 형태이다. 이때, $M$은 MUSEUM이 해당 프로그램에 대해 생성한 모든 변이들을 뜻한다. $f(s) \cap \overline{p_m(s)}$은 원본 프로그램에서 구문 $s$를 실행(커버)하였고 실패한 테스트 케이스이면서 동시에 변이 $m$에서는 $s$를 도달하지 않고 성공한 테스트 케이스의 집합이다. 즉, $\Sigma_{m \in M} |f(s) \cap \overline{p_m(s)}|$는 모든 변이에 있어서 테스트 결과가 실패에서 성공으로 변경되었을 때, 실패에서는 구문 $s$를 실행하였으나 성공하는 경우(변이)에서는 실행하지 않는 경우의 수이다. $cov_f 2nonc ov_p$는 모든 변이를 아울러, 테스트 결과가 실패에서 성공으로 바뀜에 따라서 구문이 실행되었던 구문이 실행되지 않는 경우의 총합을 나타낸다.

$$susp_{Met+}(s) = susp_{Met}(s) + \frac{1}{\Sigma_{m \in M_{\max}} |f \cap k_m|} \times \frac{\Sigma_{m \in M_{\max}} |f(s) \cap \overline{k_m(s)}|}{cov_f 2nonc ov_{kill}} \quad (4)$$

식 (4)는 Metallaxis의 결함 의심도 계산식 식 (2)에 새로운 항을 추가한 것이다. 해당 식에서 $M_{\max}$는 Metallaxis 식을 사용했을 때 구문 별로 가장 높은 의심도를 갖게 하는 변이들의 집합을 나타내며 $\Sigma_{m \in M_{\max}} |f \cap k_m|$은 원본 프로그램에서 실패하던 테스트 케이스가 구문에서 가장 높은 의심도를 갖는 변이를 kill하는 모든 경우의 수를 가리킨다. $f(s) \cap \overline{k_m(s)}$는 원본 프로그램에서 구문 $s$를 실행하였고 실패한 테스트 케이스이면서 동시에 변이 $m$에서는 구문 $s$를 실행하지 않고 변이 $m$을 kill하는 테스트 케이스의 집합이다. 따라서, $\Sigma_{m \in M_{\max}} |f(s) \cap \overline{k_m(s)}|$은 실패하는 테스트 케이스가 모든 구문에서 생성된 가장 높은 의심도를 갖는 변이를 kill 함에 따라, 원본 프로그램에서는 구문 $s$를 실행하였으나 변이 프로그램에서는 실행하지 않는 경우의 수를 나타낸다. $cov_f 2nonc ov_{kill}$는 Metallaxis 식을 사용했을 때 구문 별로 가장 높은 의심도를 갖게 하는 변이들에 대해서, 원본 프로그램에서 실패하던 테스트 케이스가 변이를 kill함에 따라서 구문이 실행되다가 실행되지 않는 경우의 총 합을 나타낸다.

식 (3)과 식 (4)의 결함 의심도 계산식은 해당 구문으로부터 생성된 변이가 아닌 경우에 대해서도, 테스트 케이스가 변이를 실행할 때 변경되는 커버리지 정보를 바탕으로 관련 구문에 의심도를 상향 조정함으로써, 변이와 구문 간의 일대일 관계로 한정되어 있는 기존의 MBFL 방식보다 변이 정보를 포괄적으로 사용하도록 하여, 결함 위치에서 결함과 직결된 변이 생성이 어려운 코드 누락 결함의 결함 위치 추정을 향상시키는 효과를 기대할 수 있다.

### 3.2 예제(1) - MUSEUM+을 통한 정확도 향상

MUSEUM+는 2.2장의 예제에 대해 결함 위치를 정확히 추정할 수 있다. 그림 4(a)는 실패에서 성공으로 테스트 결과가 변경된 두 변이 M3와 M5에서 TC5가 실행한 구문을 나타낸다. 검은색 동그라미(●)는 변이 프로그램이 실행시킨 구문을 나타내며 흰색 동그라미(○)는 기존 프로그램의 테스트 케이스에서 실행하고 해당 변이 프로그램에서의 테스트 케이스에서는 실행하지 않은 구문을 가리킨다.

프로그램에서의 테스트 케이스 커버리지와 두 변이 프로그램의 커버리지를 비교해 보았을 때 두 변이 프로그램 모두에서 4번 구문과 6번 구문을 실행하지 않았다. 따라서 4번 구문과 6번 구문은 M3와 M5의 결과를 바탕으로 결함 의심도를 계상한다. 그림 4(b)는 MUSEUM+으로 계산한 각 구문의 결함 의심도를 나타낸다. MUSEUM+는 4번 구문, 즉 코드 누락 직전 행에 가장 높은 의심도를 부여하여 결함 위치를 정확히 탐지했다.

| M3 | TC5 (x=2) |
|---|---|
| 2: int div=x; | ● |
| 3: if(x%2!=0){//mut | ● |
| 4: div= 2; | ○ |
| 6: div = 6; } | ○ |
| 7: else if(x%3==0){ | ● |
| 8: div = 3; } | |
| 9: return div; } | ● |
| Test result | F→P |

| M5 | TC5 (x=2) |
|---|---|
| 2: int div=x; | ● |
| 3: if(false){ //mut. | ● |
| 4: div= 2; | ○ |
| 6: div = 6; } | ○ |
| 7: else if(x%3==0){ | ● |
| 8: div = 3; } | |
| 9: return div; } | ● |
| Test result | F→P |

(a) Coverage of TC5 on M3 and M5

| Statements | $\lvert p(s) \cap f_m \rvert$ | $\lvert f(s) \cap f_m \rvert$ | $\lvert f(s) \cap \overline{p_m(s)} \rvert$ | Susp | Rank |
|---|---|---|---|---|---|
| 2:int div=x; | 2 | 0 | 0 | -0.04 | 4 |
| 3:if((x % 2==0){ | 8 | 2 | 0 | 0.23 | 2 |
| 4: div= 2; | 0 | 0 | 2 | 0.25 | 1 |
| 6: div = 6; } | 2 | 0 | 2 | 0.21 | 3 |
| 7:else if(x%3==0){ | - | - | - | - | - |
| 8: div = 3; } | - | - | - | - | - |
| 9:return div; } | 8 | 0 | 0 | -0.16 | 5 |

(b) Suspiciousness Score of MUSEUM+

그림 4 예제 프로그램에 대한 MUSEUM+ 결과

Fig. 4 The result of the MUSEUM+ on the example program

### 3.3 예제(1) - MUSEUM+을 통한 정확도 향상

그림 5는 2.2장의 예제에 대해서 Metallaxis+으로 계산한 각 구문의 의심도를 보인다. 그림 5(a)는 구문 별로 가장 높은 의심도를 갖는 변이에 대해서 해당 변이를 kill하고 원본에서 실패하는 테스트케이스의 커버리지 변화를 보인다. 예제에서 구문 별로 원본에서 실패하고 변이를 kill하는 테스트 케이스가 존재하는, 가장 높은 의심도를 갖는 변이는 구문 3, 6, 9에서 생성된 M5, M8(또는 M9), M10(또는 M11)이다. 그림 5(a)에서 보이듯, M8과 M10에서의 테스트 케이스 출력은 변하지만 커버리지는 변하지 않는다. 하지만 원본 프로그램에서 실행되던 4번 구문과 6번구문이 변이 M5에서는 실행되지 않았다. 따라서 Metallaxis+에서는 구문 4번과 6번 의심도에 점수를 부여한다. Metallaxis+를 사용하여 구문 별로 점수를 계산한 것은 그림 5(b)에서 보인다. Metallaxis+는 구문 3과 구문 6중에 구문 6에 점수를 더 주어 결함 위치를 정확히 탐지하였다.

## 4. 실 험

### 4.1 연구 질문

본 연구에서 제안한 커버리지 변화정보를 고려한 뮤테이션 기법의 효과성을 평가하기 위해서 세가지 연구 질문에 대한 실험을 진행한다.

- RQ1. 코드 누락 결함에 대해서 MUSEUM+와 Metallaxis+가 기존 뮤테이션 기반 결함 위치 추적 기법, MUSEUM과 Metallaxis의 정확도를 얼마큼 향상하였는가?

| M5 | TC5 (x=2) |
|---|---|
| 2: int div=x; | ● |
| 3: if(false){//mut | ● |
| 4: div= 2; | ○ |
| 6: div = 6; } | ○ |
| 7:else if(x%3==0){ | ● |
| 8: div = 3; } | |
| 9: return div; } | ● |
| Test Output | 6→2 |

| M8 | TC5 (x=2) |
|---|---|
| 2: int div=x; | ● |
| 3: if(false){//mut | ● |
| 4: div= 2; | ● |
| 6: div = 6; } | ● |
| 7:else if(x%3==0){ | |
| 8: div = 3; } | |
| 9: return div; } | ● |
| Test Output | 6→2 |

| M10 | TC5 (x=2) |
|---|---|
| 2: int div=x; | ● |
| 3: if(false){//mut | ● |
| 4: div= 2; | ● |
| 6: div = 6; } | ● |
| 7:else if(x%3==0){ | |
| 8: div = 3; } | |
| 9: return div; } | ● |
| Test Output | 6→0 |

(a) Coverage of TC5 on M3, M5 and M8

| Statements | $\lvert p(s) \cap f_m \rvert$ | $\lvert f(s) \cap f_m \rvert$ | $\lvert f(s) \cap \overline{p_m(s)} \rvert$ | Susp | Rank |
|---|---|---|---|---|---|
| 2:int div=x; | 2 | 0 | 0 | -0.04 | 4 |
| 3:if((x % 2==0){ | 8 | 2 | 0 | 0.23 | 2 |
| 4: div= 2; | 0 | 0 | 2 | 0.25 | 1 |
| 6: div = 6; } | 2 | 0 | 2 | 0.21 | 3 |
| 7:else if(x%3==0){ | - | - | - | - | - |
| 8: div = 3; } | - | - | - | - | - |
| 9:return div; } | 8 | 0 | 0 | -0.16 | 5 |

(b) Suspiciousness Score of Metallaxis+

그림 5 예제 프로그램에 대한 Metallaxis+ 결과

Fig. 5 The result of the Metallaxis+ on the example program

- RQ2. 코드 누락 결함이 아닌 결함에 대해서 MUSEUM+, Metallaxis+과 MUSEUM, Metallaxis사이에 어떠한 성능 변화가 있었는가?
- RQ3. 스펙트럼 기반 결함 위치 추적 기법(SBFL)에 대해서 MUSEUM, Metallaxis과 MUSEUM+, Metallaxis+를 비교했을 때 얼마큼 결함을 정확하게 추정하는가?

### 4.2 실험 설정 및 수행

MUSEUM+와 Metallaxis+을 실험으로 위의 연구 질문들에 대하여 평가하기 위하여 Defects4J[5][6]의 JFreeChart[7] 결함 16개를 실험에 사용하였다(표 1). 이 중 10개 결함은 코드 누락 결함으로 JFreeChart에 있는 코드 누락 결함 전체를 사용하였다(표 1(a), 표 1(c)). 코드 누락 결함에 더하여, 일반적인 결함에서의 성능도 확인하기 위해 6개의 코드 누락이 아닌 결함을 임의로 선정하여 실험에 사용했다(표 1(b), 표 1(d)). MUSEUM은 실패하는 테스트 케이스를 하나만 사용하기 때문에 버그와 관련된 모든 실패하는 테스트 케이스를 사용하는 Metallaxis와 사용하는 테스트케이스 집합과 변이의 개수가 다르다. 표 1(a)와 표 1(b)는 MUSEUM에서 사용된 테스트 케이스와 변이의 개수를 보이며 표 1(c)와 표 1(d)는 Metallaxis에서 사용된 테스트 케이스와 변이의 개수를 보인다. 선택한 16개 결함 프로그램은 Defects4J 데이터를 바탕으로 하되, 이를 면밀히 재검토하여 잘못된 설정은 수정하여 사용하였다.

실험은 Defects4J 환경을 기본으로 사용하되, 변이 생성에는 Major 1.3.4 [11,12]를 사용하였다.

Defects4J에서 선택한16개 결함 프로그램 중 6개 결함에는 실패하는 테스트 케이스가 2개 이상 존재하였는데, MUSEM과 MUSEUM+에서는 이 중 하나의 테스트 케이스를 임의로 정해 사용하였고 Metallaxis와 Metallaxis+에서는 실제 결함을 드러내는 모든 테스트케이스를 사용하였다. Defejcts4J가 제공하는 테스트 케이스들 중 실패하는 테스트케이스가 실행하는 구문들 중 하나 이상의 구문을 실행하는 테스트 케이스을 실험에 사용하였다. RQ3에서 SBFL기법을 계산하기 위해 사용한 테스트 케이스는 Metallaxis에서 사용된 테스트 케이스와 같다.

또한, 총 16개 결함 사례 중 13건의 경우, 실패하는 테스트케이스는 결함과 관련 없는 구문을 지우는 정화 작업(test case purification)[13]을 수행하여 사용하였다. SBFL 기법, MUSEUM+ 그리고 Metallaxis+에서 사용하는 커버러지 정보는 Gzoltar[14,15]를 이용하여 구하였다.

실험은 Intel Core i5-6600 @3.30GHz의 CPU, 8GB 메모리, Ubuntu 16.04.6 LTS 환경에서 진행되었다. 실험에 사용된 16개의 결함 프로그램에 대해서, MUSEUM과 MUSEUM+에는 평균1191개의 변이가, Metallaxis와 Metallaxis+에는 평균 1964개의 변이가 생성되어 사용되었다.

표 1 실험에 사용된 16개 Defects4J 결함 사례
Table 1 The Defects4J artifacts used in the experiment

(a) MUSEUM:Omission fault

| Study Object | #TC | # mutants |
|---|---|---|
| Chart-2 | 838 | 212 |
| Chart-3 | 570 | 269 |
| Chart-4 | 1219 | 1325 |
| Chart-5 | 303 | 110 |
| Chart-14 | 978 | 715 |
| Chart-15 | 1054 | 2025 |
| Chart-19 | 978 | 1645 |
| Chart-21 | 582 | 211 |
| Chart-25 | 1149 | 7104 |
| Chart-26 | 1065 | 3122 |

(b) MUSEUM:Non-omission fault

| Study Object | #TC | # mutants |
|---|---|---|
| Chart-1 | 1300 | 1218 |
| Chart-7 | 525 | 287 |
| Chart-11 | 264 | 57 |
| Chart-13 | 480 | 499 |
| Chart-17 | 498 | 222 |
| Chart-24 | 10 | 34 |

(c) Metallaxis:Omission fault

| Study Object | #TC | # mutants |
|---|---|---|
| Chart-2 | 838 | 212 |
| Chart-3 | 570 | 269 |
| Chart-4 | 1737 | 4930 |
| Chart-5 | 303 | 110 |
| Chart-14 | 978 | 715 |
| Chart-15 | 1054 | 2025 |
| Chart-19 | 978 | 1645 |
| Chart-21 | 582 | 211 |
| Chart-25 | 1149 | 7104 |
| Chart-26 | 1443 | 11886 |

(d) Metallaxis:Non-omission fault

| Study Object | #TC | # mutants |
|---|---|---|
| Chart-1 | 1300 | 1218 |
| Chart-7 | 525 | 287 |
| Chart-11 | 264 | 57 |
| Chart-13 | 480 | 499 |
| Chart-17 | 498 | 222 |
| Chart-24 | 10 | 34 |

### 4.3 실험 결과

전체적으로 MUSEUM의 평균, 최소, 최대의 등수가 203.6, 1, 1185인 반면 MUSEUM+의 평균, 최소, 최대의 등수가 74, 1, 549.5로 현격하게 향상되었다. 또한 Metallaxis의 평균, 최소, 최대의 등수가 162.9, 1.5, 1072인 반면 Metallaxis+의 평균, 최소, 최대의 등수가 38.5, 1, 166으로 등수 향상이 된 것을 확인할 수 있다.

- RQ1. 코드 누락 결함에 대한 제안한 기법의 효과성

코드 누락 결함에 대해서 제안한 기법이 기존 MBFL 기법들보다 높은 정확도로 결함 위치를 추정한다. 표 2는 10개의 코드 누락 결함에 대해서 기존 MBFL기법인 MUSEUM과 Metallaix, 각 기법을 확장시킨 MUSEUM+와 Metallaxis+, 그리고 SBFL 기법인 Dstar[16], Op2[17], 그리고 Ochiai[18]를 통해 추정한 결함 구문의 결함 의심도 중 가장 상위 등수의 값(i.e., Expense Metric)을 보인다.

MUSEUM+는 코드 누락 결함 중 총 6개 경우에서 MUSEUM보다 향상된 정확도를 보였으며(*표시), 나머지 4개에 대해서도 등수가 2.0 미만의 근소한 차이를 보이며 코드 누락 결함에 대해 MUSEUM의 성능 향상을

표 2 코드 누락 결함에 대한 실험 결과
Table 2 The experiment result of the omission faults

| Study object | MSM | MSM+ | Met | Met+ | DStar | Op2 | Ochiai |
|---|---|---|---|---|---|---|---|
| C2 | 81.5 | 19.5* | 9 | **8.5*** | 36 | 36 | 36 |
| C3 | 67 | 69 | 6.5 | **3*** | 6.5 | 6.5 | 6.5 |
| C4 | 593 | 549.5* | 141 | 141 | 49 | **16** | 55 |
| C5 | 13 | 13 | **3** | **3** | 7.5 | 7.5 | 7.5 |
| C14 | 1.5 | **1*** | 5 | **1*** | 4.5 | 4.5 | 4.5 |
| C15 | 941 | 36* | 686.5 | 85* | **32** | **32** | **32** |
| C19 | 275 | 275 | 535 | 166* | **3** | **3** | **3** |
| C21 | 27.5 | 28 | **11** | 12.5 | 21.5 | 21.5 | 21.5 |
| C25 | 1185 | 122* | 1072 | 58.5* | **31.5** | **31.5** | **31.5** |
| C26 | 4.5 | **11*** | 7 | 13 | 137 | 137 | 137 |

달성함을 알 수 있다. 특히, MUSEUM이 낮은 성능을 보인 4개에 대해 정확도를 향상했으며, 그 중 3개에 대해(Chart-2, Chart-15, Chart-25) 결함 위치의 등수를 현격히 단축했다(각각 76%, 96%, 90% 단축).

Metallaxis+는 10 개의 코드 누락 결함 중 6개의 결함에 대해Metallaxis보다 높은 정확도를 보였으며(*표시) 나머지 4개의 결함에는 등수 5 이내의 등수 차이를 보였다. Metallaxis에서 성능이 낮았던 3개의 결함(Chart-15, Chart-19, Chart-25)에 대해서 등수가 현저히 향상되는 것을 확인할 수 있다(각각 87%, 69%, 94% 단축).

- RQ2. 코드 누락 결함이 아닌 결함에 대한 제안한 기법의 효과성

일반 결함에 대해서 제안한 기법과 기존 MBFL기법는 근소한 차이를 보인다. 표 3은 6개의 일반적 결함에 대해서 기존 MBFL기법인 MUSEUM과 Metallaix, 각 기법을 확장시킨 MUSEUM+와 Metallaxis+, 그리고 SBFL 기법인 Dstar, Op2, 그리고 Ochiai를 통해 추정한 결함 구문의 결함 의심도 중 가장 상위 등수의 값(i.e., Expense Metric)을 보인다.

일반적 결함의 경우, MUSEUM+는 1개(Chart-13) 결함에 있어서 MUSEUM보다 좋은 성능을 보였고, 나머지 4개에 있어서는 동일하거나 등수 1 수준의 근소한 차이만을 보였다. 따라서, MUSEUM+가 MUSEUM에 비하여 총체적으로 성능 향상을 보임을 실험을 통하여 확인할 수 있었다.

표 3 일반 결함에 대한 실험 결과
Table 3 The experiment result of the non-omission faults

| Study object | MSM | MSM+ | Met | Met+ | DStar | Op2 | Ochiai |
|---|---|---|---|---|---|---|---|
| C1 | 2 | 2 | **1.5** | 2 | 35 | 35 | 35 |
| C7 | **25** | **25** | 83.5 | 105 | 34.5 | 34.5 | 34.5 |
| C11 | 18.5 | 18.5 | **8.5** | 11.5 | 10 | 10 | 10 |
| C13 | 21.5 | 20.5* | 33.5 | **1*** | 35 | 35 | 35 |
| C17 | **1** | 2 | 2.5 | 3 | 2 | 2 | 2 |
| C24 | **1.5** | 2.5 | 2 | 2 | 3 | 3 | 3 |

Metallaxis+는 일반적 결함 중 1개의 결함(Chart-13)에 대해서 Metallaxis보다 높은 정확도를 보였다. 4개의 결함에 대해서는 낮은 정확도를 보였는데 이중 3개의 결함에 대해서는 등수 3이하의 차이를 보인 반면 1개의 결함에 대해서는 등수가 21.5 차이를 보인다.

- RQ3. SBFL 결과의 기준으로 기존 MBFL 기법과 제안한 기법의 효과성 비교

MBFL 기법은 코드 누락 결함에 대해서 SBFL 보다 부정확하게 버그 위치를 추정한다. 총 10개의 코드 누락 결함 중 MUSEUM은 2개(Chart-14, Chart-26) 결함에 대해서만, Metallaxis는 4개(Chart-2, Chart-5, Chart-21, Chart-26) 결함에 대해서 SBFL보다 높은 정확도를 보인다. 각 기법이 추정하는 버그 위치의 평균 등수를 비교한 것은 다음과 같다. 10개의 코드 누락 결함에서 버그 위치를 MUSEUM은 평균 318.9 등수로, Metallaxis는 평균 247.6 등수로 추정한다. SBFL기법 중 버그 위치의 평균 등수가 가장 높은 Op2의 경우 버그 위치를 평균 29.55 등수로 추정한다. Op2와 MUSEUM의 평균 등수 차는 289.35, Op2와 Metallaxis의 평균 등수 차는 218.05가 난다. SBFL의 다른 두 기법들의 평균 등수와 Op2의 평균 등수의 차는 5등수 이내임으로 MBFL과 SBFL 사이의 평균 등수 차이가 200이상 나는 것을 보인다.

코드 누락 결함에 대해 제안한 MBFL 기법과 SBFL 기법의 성능 결과 변화와 기존 MBFL 기법과 SBFL 기법의 성능 결과 변화를 비교했을 때 제안한 기법이 보다 정확하게 버그 위치를 추정한다. MUSEUM+는 3개의 버그에 대해서 SBFL보다 정확하게 버그 위치를 추정한다. 이는 MUSEUM이 SBFL보다 정확하게 버그 위치를 추정한 2개의 결함과 더불어 1개의 결함(Chart-2)에 대해서 결함 위치의 등수가 현격히 올라가면서 SBFL보다 정확히 추정한 것이다. Metallaxis+는 6개의 버그에 대해서 SBFL보다 정확하게 버그 위치를 추정한다. 해당 6개의 결함은 Metallaxis가 SBFL보다 정확하게 버그 위치를 추정하는 4개의 결함에 2개의 결함(Chart-3, Chart-14)위치의 등수가 상향됨에 따른 것이다. 10개의 코드 누락 결함의 평균 등수에서도 결과의 향상을 보인다. 10개의 코드 누락에 대해서 MUSEUM+은 평균 112.4 등수를 보이며, Metallaxis+은 평균 49.15 등수를 보인다. Op2와 MUSEUM+ 평균 차는 82.85, Op2와 Metallaxis+의 평균 등수 차는 19.6이 난다. Op2와 평균 등수 차이가 200이상 나는 기존 MBFL 기법과 비교했을 때, 제안한 MBFL 기법이 SBFL 기법 결과를 비교함에 있어서도 기존 MBFL기법 보다 코드

누락 결함에 있어서 등수 향상이 있다는 것을 보인다.

코드 누락이 아닌 일반 결함 위치 추정에 대해서, 기존 MBFL기법은 SBFL기법보다 높은 정확도를 보인다. 6개의 일반 결함에 대해서 MUSEUM은 모든 SBFL기법보다 5개의 결함에 대해서 위치를 정확하게 추정하며, Metallaxis의 경우 4개의 결함에 대해서 모든 SBFL기법보다 위치를 정확하게 추정한다.

일반 결함에 대해 제안한 MBFL 기법과 SBFL의 성능 결과 변화와 기존 MBFL 기법과 SBFL 기법의 성능 결과 변화를 비교했을 때 그 차이는 근소하다. MUSEUM+의 경우 5개의 결함에 대해서 SBFL보다 높은 정확도를 보였으며, 해당 5개의 결함은 MUSEUM이 SBFL보다 높은 정확도를 보인 5개의 결함과 일치한다. Metallaxis+는 3개의 결함에 대해서 SBFL기법보다 높은 정확도를 보였다. 해당 3개의 결함은 Metallaxis가 SBFL 기법보다 높은 정확도를 보인 4개의 결함 중 3의 결함과 일치한다. 나머지 1개의 결함(Chart-11)에서는 Metallaxis에서 Metallaxis+의 점수가 3 등수 증가했기 때문에 SBFL기법 보다 낮은 정확도를 보였으나 등수 차이가 근소하다는 것을 알 수 있다.

### 4.4 실험 타당성 위협 요소(Threats to validity)

해당 실험의 외적 타당성 위협 요소(external threat)로 실험 대상 프로그램의 제한성을 들 수 있다. 본 연구는 이 제한성의 한계를 극복하기 위해 여러 테스팅 연구에서 널리 사용되고 있는 Defects4J 벤치마크에서 실험 대상을 선별하여 관련된 연구와 실험 결과 비교가 용이하게 하였으며, JFreeChart의 26개 결함 경우 중 코드 누락(omission fault)에 해당하는 10개 결함 전수를 실험 대상에 포함시켰다. 또한, 코드 누락 결함 이의의 결함 중에서도 다양한 사례를 추가하여 총 16개의 결함 사례를 대상으로 한 실험을 설계하였다. 본 연구에는 JFreeChart 내에 다양한 결함을 비교하는데 집중함으로써 다양한 프로젝트로부터 결함을 수집하였을 때, 각 프로젝트의 특이성이 실험에 미치는 영향이 적도록 통제하였다.

구성 타당성 위협 요소(construct threat)로 성능 평가 방식을 EXAM score에 한정하여 사용한 것을 들 수 있다. 다른 기준을 사용할 경우 평가 결과가 달라질 수 있으나, EXAM score는 관련 연구의 실험에서 널리 통용되는 평가 방식임을 고려하여 이와 같이 설계하였다. 실험 내부 타당성 위협(internal threat)으로 구현 과정에서의 오류가 있을 수 있다. 이러한 위협 요소를 극복하기 위하여 프로그램을 여러 번 검토하였을 뿐만 아니라, 각 실험에서 수행된 뮤테이션 테스팅 결과에 대한 정성적인 분석을 통해 실험 결과에 오류가 없음을 확인하는 과정을 거쳤다.

## 5. 결 론

본 논문은 코드 누락 결함에 대해 뮤테이션 기반 결함 위치 추정 기법이 갖는 한계점을 효과적으로 극복하기 위해 변이 프로그램에서의 테스트 케이스 커버리지 정보를 결함 의심도 계산에 활용하는 방법을 제안한다. 제안한 기법을 기존 MUSEUM 기법과 Metallaxis기법을 확장하여 MUSEUM+, Metallaxis+로 구현한 후 16개 Defects4J 내 JFreeChart 결함에 적용한 결과, 총 10개의 코드 누락 결함 중 MUSEUM+과 Metallaxis+는 6개에 대해 정확도를 향상시켰으며, 두 기법 모두 16개 전반에 있어 MUSEUM과 Metallaxis보다 정확도가 향상됨을 확인할 수 있었다.

## References


[1] Y. Lin, J. Sun, L. Tran, G. Bai, H. Wang, and J. Dong, "Break the Dead End of Dynamic Slicing: Localizing Data and Control Omission Bug," *Proc. of IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2018.

[2] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A Survey on Software Fault Localization," *IEEE Transactions on Software Engineering (TSE)*, Vol. 42, No. 8, pp. 707-740, 2016.

[3] S. Hong, T. Kwak, B. Lee, Y. Jeon, B. Ko, Y. Kim, and M. Kim, "MUSEUM: Debugging Real-world Multilingual Programs Using Mutation Analysis," *Information and Software Technology (IST)*, No. 82, pp. 80-95, 2017.

[4] M. Papadakis and Y. L. Traon, "Metallaxis-FL: mutation-based fault localization," *Journal of Software Testing, Verification and Reliability (STVR)*, Vol. 25, pp. 605-628, 2015.

[5] R. Just, D. Jalali, and M. D. Ernst, "Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs," *Proc. of the International Symposium on Software Testing and Analysis (ISSTA)*, pp. 437-440, 2014.

[6] Defects4J, [Online]. Available: https://github.com/rjust/defects4j

[7] JFreeChart, [Online]. Available: https://jfree.org/jfreechart

[8] Y. Kim and S. Hong, "DEMINER: Test Generation for High Test Coverage through Mutant Exploration," *Software Testing, Verification and Reliability (STVR)*, online published, 28 Oct. 2019.

[9] H. Peng, Y. Shoshitaishvili, and M. Payer, "T-Fuzz: Fuzzing by Program Transformation," *Proc. of IEEE Symposium on Security and Privacy (SP)*, 2018.

[10] J. Holmes and A. Groce, "Using Mutants to Help Developers Distinguish and Debug (Compiler) Faults," *Software Testing, Verification and Reliability (STVR)*,

to appear.
[11] R. Just, "The Major Mutation Framework: Efficient and Scalable Mutation Analysis for Java," *Proc. of the International Symposium on Software Testing and Analysis (ISSTA)*, 2014.
[12] The Major Mutation Framework, [Online]. Available: https://mutation-testing.org
[13] J. Xuan and M. Monperrus, "Test Case Purification for Improving Fault Localization," *Proc. of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, 2014.
[14] J. Compos, A. Riboira, A. Perez, and R Abreu, "Gzoltar: an eclipse plug-in for testing and debugging," *Proc. of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2012.
[15] Gzoltar, [Online]. Available: https://github.com/GZoltar/gzoltar
[16] W.E. Wong, V. Debroy, R. Gao, and Y. Li, "The DStar method for effective software fault localization," *IEEE Transaction on Reliability*, Vol. 63, pp. 290-308, 2014.
[17] L. Naish, H. J. Lee, and K. Ramamohanarao, "A model for spectra-based software diagnosis," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 20, pp. 11, 2011.
[18] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. C. Van Gemund, "On the accuracy of spectrum-based fault localization," *Proc. of Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION (TAIC PART-MUTATION)*, 2007.

전 주 영
2019년 한동대학교 전산전자공학부(학사). 2019년~현재 한동대학교 일반대학원 정보통신공학과 석사과정. 관심분야는 소프트웨어 자동 디버깅, 뮤테이션 테스팅

홍 신
2007년 KAIST 전산학부(학사). 2010년 KAIST 전산학부(석사). 2015년 KAIST 전산학부(박사). 2016년~현재 한동대학교 전산전자공학부 조교수. 관심분야는 소프트웨어 자동 테스팅 및 디버깅