# 인공신경망을 활용한 소프트웨어 결함 위치 추정 기법[1]

조준혁, 이지현, 자프르 아만
전북대학교 소프트웨어공학과
e-mail : bionicle92@jbnu.ac.kr

# Software Fault Localization using Artificial Neural Network

Jun-Hyuk Jo, Jihyun Lee, Aman Jaffari
Dept. of Software Engineering, Chonbuk National University

**요 약**

소프트웨어 시험 후 발견된 결함을 제거하기 위해서는 먼저 해당 결함의 위치를 정확히 찾아야 한다. 결함의 위치를 찾는 작업은 많은 양의 소스코드를 검토해야 하기 때문에 많은 노력을 요구한다. 해당 노력을 줄이기 위해 슬라이싱 기법, 스펙트럼 기법, 모델 기반 기법 등 많은 기법들이 연구되었다. 하지만 이들 연구들은 결함 위치로 추정한 탐색 영역의 범위가 넓어 결과의 효과가 떨어지는 단점이 있다. 그래서 결함 위치 추정의 정확도를 높이고 결함 위치 파악의 효과를 높이기 위해 본 논문은 프로그램 소스 코드 문장에 대한 시험케이스의 커버리지 정보, 시험의 Pass/Fail 여부, Define-Use 의 관계에 있는 문장 정보를 활용하여 각 문장의 결함 의심도를 산출하는 방법을 제안한다. 제안 방법을 실험을 통하여 확인한 결과, 낮은 지역화 비용으로 결함 위치 추정을 할 수 있었다.

## 1. 서론

결함 위치추정 (fault localization)은 프로그램에서 결함 (fault)이 있을 것으로 예상되는 코드들을 개발자에게 제시하는 기법이다. 결함 위치추정은 일반적으로 개발자가 실패가 발생한 프로그램의 실행을 일일이 추적하여 결함을 찾아야 하기 때문에 많은 시간과 비용이 요구된다. 따라서 이런 노력을 줄이기 위해 결함 위치 추정 기법들이 제안되었다. W.E. Wong 은 소프트웨어 결함 위치 식별 기법들을 슬라이싱 (Slicing), 스펙트럼 기반 (spectra-based), 프로그램 상태 기반 (program state-based), 머신러닝 기반 (machine learning-based), 모델 기반 (model-based) 기법 등으로 분류하여 분석하였다[1]. 스펙트럼 기법은 조건부 분기와 같은 특정 관점에서 프로그램 실행 정보를 설명한다. 실행이 실패하면 이러한 정보를 사용하여 시험 중에 프로그램의 어느 부분이 실행 중에 다루어 졌는지 나타내고, 이를 통해 의심스러운 코드를 식별할 수 있다. 머신러닝을 기반으로 한 결함 위치추정 기법은 시험케이스 (test case)의 커버리지 정보와 실행 결과를 수집하여 기계학습 모델을 통해 이들 간의 관계를 학습하고, 문장에 결함이 있을 확률을 출력한다. 이들 결함 위치 추정 기법 중 기본이 되는 위치 추정 기법은 프로그램 슬라이싱 기법이다. 슬라이싱 기반 결함 위치 추정 기법[2]은 시험케이스 실패 관련 문장들만으로 결함의 탐색 영역을 제한한다.

하지만 해당 기법의 결과는 결함이 시험케이스가 실패한 경우의 영역이 아닌 다른 부분에 존재하는 일도 많고, 또한 해당 영역에 존재한다 하더라도 실패한 시험케이스들을 기준으로 하기 때문에 결과 슬라이스의 크기가 커서 위치 추정의 공헌도가 떨어지는 단점이 있다.

본 논문에서는 이 문제를 해결하기 위해 인공신경망 모델 (Neural Network model)을 활용한다. 인공신경망 모델은 입력 데이터에 따라 특성 혹은 패턴 파악을 통한 데이터 분류 (classification) 또는 군집화 (clustering)가 가능하다. 해당 모델의 이러한 특성을 활용하여, 하나하나의 문장 단위로 각 문장의 특성을 입력데이터로 하여 문장 별 결함 의심도를 산출한다. 우선적으로 프로그램에서 각 문장에 대한 시험케이스의 커버리지 정보, Pass/Fail 정보를 추출하고, Define-Use (이하 DU)에 해당하는 문장들을 분류한다. 그런 다음 커버리지 정보와 시험케이스 Pass/Fail 여부, DU 정보를 기준으로 문장의 가중치를 결정하는 전처리 과정을 거친다. 전처리 결과를 입력으로 인공신경망 모델의 가중치를 조정하여 최종적으로 각 문장의 결함 의심도를 산출한다. 산출된 결과는 각 문장 별 결함 의심도를 나타내기 때문에 임의의 슬라이스 단위가 아니라 하나하나의 문장단위로 결함이 의심되는 범위를 좁힐 수 있다.

[1] 이 논문은 2018 년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2017R1D1A3B03028609)

## 2. 관련 연구

개발자의 결함 수정 노력을 줄이기 위한 결함 위치 추정 기법들 중 CBFL (coverage based fault localization)은 프로그램 코드의 의심도를 측정하는 방법을 개선한 연구이다[2]. 프로그램 커버리지를 이용하여 결함으로 의심되는 코드들을 개발자에게 의심도 순으로 자동 제시하여 결함 확률이 보다 높은 코드를 확인할 수 있다. 하지만 CBFL 기법에서 사용되는 테스트케이스 중 결함을 실행했음에도 실패하지 않는 시험케이스의 수가 많을 경우, 결함 코드가 그렇지 않은 코드들보다 낮은 의심도를 가지는 문제점이 있다. 이와 같은 문제를 해결한 방법이 범위 기반 접근 방식(lightweight fault-localization)이다[3]. 이 방법은 결함 위치 추정의 대표적 요소가 되는 소스 코드 상의 문장, 분기, DU-쌍들을 복합적으로 고려하여 결함 위치 추정의 정확도를 보다 향상시켰다. 하지만 이 연구에서는 런타임 오버헤드가 높게 일어나는 문제를 가진다.

P. Zhang 등은 JSlice 도구를 사용한 동적 슬라이싱을 통해 실행가능한 코드만을 추출하여 테스트를 진행했을 때 더 높은 정확도로 오류를 찾을 수 있음을 검증하였다[4]. 또한 A. Perez 등은 실패한 시험케이스가 있는 경우 '스펙트럼 기반 접근법'을 사용하여 결함을 감지하는 시험스위트의 진단 기능을 평가하는 새로운 측정 기준을 제안하였다. 이 방법은 시험에서 사용한 시험케이스들을 결함을 검증하는 평가 척도로 활용하여 각 시험케이스 별 결함 검출률의 정확도를 높였다[5].

본 논문에서 제안하는 방법은 인공신경망과 관련된 선행연구에서 사용한 기법을 기반으로 개발했다. W.E.Wong 등의 역전파 신경망 방법 (Back-Propagation neural network-based method, BP 신경망)[6]을 시작으로 관련 연구가 진행되어 왔다. 프로그램 커버리지 정보를 인공신경망을 통해 학습시키고, 이를 통해 가상의 각 문장만을 지나는 프로그램 커버리지 정보의 결과값을 예측한다. 이 결과값을 의심도로 활용하는 것이 인공신경망을 활용한 결함 위치 식별 기법의 기본이다. BP 신경망을 기반으로 한 결함 위치 추정 기법은 시험케이스의 커버리지 데이터와 해당 실행 결과가 수집되어 네트워크가 BP 신경망을 학습하여 이들 간의 관계를 학습하고, 문장 별로 결함이 있을 가능성을 출력한다. 하지만 BP 신경망은 다중 결함 검출을 위해서는 실패한 시험케이스를 군집화 해야 하고 해당 군집화가 적합한 것인지 검증하기가 어려운 문제가 있다.

앞선 연구들에서는 시험하고자 하는 코드 조각(fragment) 변경과 새로운 평가 척도 제안, BP 신경망 모델링을 통한 학습 적용 등으로 단순 정확도는 향상되었지만 서로 다른 결함의 유형에 따른 의심도 측정과 결과를 알 수 없다. 기존의 결함 위치추정 기법들에서 공헌도가 낮고 결함 유형에 따른 결과를 알 수 없는 문제를 해결하기 위해 DU-쌍 정보를 활용하여 DU-쌍에 속하는 문장들에 가중치를 부여하는 방법을 제안한다.

## 3. 인공신경망 기반 결함 추정 기법

본 장에서는 제안 기법 적용을 위한 세부 절차를 설명하고 예제를 통해 방법이 결함 위치 추정을 수행하는 방법을 설명한다.

### 3.1 인공신경망 기반 결함 추정 절차

본 연구가 제안하는 기법은 그림 1 과 같이 4 개의 단계로 나누어진다. 각 단계의 세부 역할은 다음과 같다:

- **Step1 (정보 추출):** 테스트의 대상이 되는 시스템의 소스코드를 대상으로 하여 소스코드의 Define/Use 부분과 시험케이스 Coverage 정보, Pass/Fail 정보를 추출, 정리한다.
- **Step2 (입력 데이터 생성):** DU 관계의 문장, 커버리지 정보에 따른 문장들을 구분하여 각각의 서로 다른 경우에 해당하는 문장들에 대해 가중치를 부여하여 입력 데이터 세트를 정제한다. 여기서 각 요소에 따른 가중치는 각 사항에 해당하는 경우는 "1"을, 그

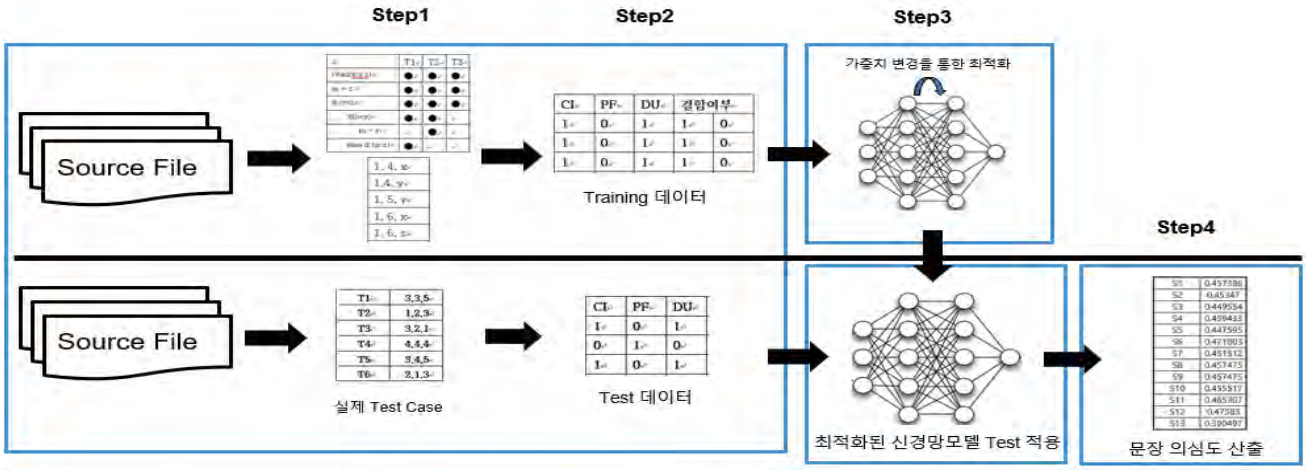


그림 1. 인공신경망 기반 결함 추정 기법의 개요

렇지 않은 경우는 “0”을 부여한다. 여기서 문장 분류 결과인 결함 문장, 비 결함 문장은 (1,0)과 (0,1)의 정수형 데이터로 변환하여 나타낸다.

• **Step3 (신경망 모델 최적화):** 만들어진 프로그램 관련 정보와 문장들 간의 관계 정보를 입력 데이터로 받아 각 문장이 실제로 결함을 가지고 있는지를 인공신경망 모델을 통해 비교 및 학습하여 신경망 모델의 가중치를 수정함으로써 모델을 최적화한다.

• **Step4 (문장 의심도 산출):** CI, PF, DU 정보만 포함된 테스트 데이터를 입력 데이터로 사용하여 최적화된 모델에 적용하고, 의심도 결과를 출력한다.

학습이 종료되고 최적화된 모델의 각 문장 분류 값들을 추출하여, 시험케이스 별로 모든 문장의 결함 및 비결함 가중치를 더하여 시험케이스의 숫자로 나누어 각 문장의 의심도를 산출한다. 최적화된 인공신경망 분류 모델을 출력하면 “[0.5511366, 0.448863]”와 같이 분류 기준이 되는 모델을 확인할 수 있다. 이 경우에는 첫 번째 항이 더 크기 때문에 “[1,0]”으로 분류되어 비 결함 문장으로 분류한다. 여기서 모델의 두 번째 항 “[0.448863]”은 특정 문장을 결함으로 분류하는 기준을 좌우하는 가중치를 나타낸다. 이를 활용해 문장의 의심도를 산출하여 출력한다.

예를 들어 첫 번째 문장 S1 에 해당하는 모델 [0.5511366, 0.448863]의 2 번째 항[0.448863]을 a 라고 할 때 이 문장의 의심도를 식으로 나타내면 다음과 같다:

$$\frac{\sum_{i=0}^{N} a_{ti}}{N}$$

단, at 는 모델의 2 번째 항, N 은 시험케이스의 수 ■

이를 통해 테스트케이스와 문장들 간의 관계를 나타내는 테이블을 작성하고 해당 데이터를 인공신경망 모델로 학습시켜 보다 각 문장들의 보다 정확한 의심도를 측정하였다.

### 3.2 예제를 통한 적용

표 1 의 예제 코드는 세 값의 변수를 입력 받아, 중간 값을 연산하여 출력하는 소스코드이다. 표는 총 13 개의 문장과 6 개의 시험케이스로 이루어져 있으며 각 문장의 커버리지와 Pass/Fail 정보를 나타낸다. 어둡게 음영처리된 열의 시험케이스는 Fail 을 의미하고, 그렇지 않은 경우는 Pass 를 나타낸다. 또한 소스코드 부분의 [du]는 DU 에 해당하는 문장을, [d]와[u]는 각각 “define”, “use”만 포함하는 문장을 나타낸다.

**표 1. 코드 정보**

| 문장 | TC1 | TC2 | TC3 | TC4 | TC5 | TC6 |
|---|---|---|---|---|---|---|
| read(x,y,z)[du] | ● | ● | ● | ● | ● | ● |
| m = z;[du] | ● | ● | ● | ● | ● | ● |
| If (y<z)[u] | ● | ● | ● | ● | ● | ● |
| If(x<y)[u] | ● | ● | | | ● | ● |
| m = y;[du] | | ● | | | | |
| else if (x<z)[u] | ● | | | | ● | ● |
| m = x;[du] | ● | | | | | ● |
| Else | | | ● | ● | | |
| If (x>y)[u] | | | ● | ● | | |
| m = y;[du] | | | ● | | | |
| else if (x>z)[u] | | | | ● | | |
| m = x;[du] | | | | | | |
| print(m)[u] | ● | ● | ● | ● | ● | ● |

이렇게 코드상에서 알아낼 수 있는 문장 커버리지와 Pass/Fail 정보, DU 에 관계에 있는 변수 정보를 인공신경망에 입력가능한 형태의 입력데이터로 변환한다.

다음 표 2 에서 PF 의 가중치는 실패한 시험케이스 내에서 커버하는 문장들에 0.5 를 부여한 것이다. 또한 DU-쌍의 가중치는 전체 문장의 DU 관계를 “(1,2,z)” 또는 “(1,3,y)”와 같이 각 변수와 문장 단위로 나타낼 때, 나타낸 결과에서 10 회 이상 포함되는 문장은 1 로 나타낸다. 또한 포함되지 않는 문장은 0, 10 회에서 1 회사이로 포함되는 문장은 각 횟수 별로 0.1 씩 더하여 계산한다. CI 는 문장 커버리지 정보(coverage information), PF 는 Pass/Fail 을, DU 는 Define-Use 쌍을 나타낸다. 나머지 우측의 컬럼은 값이 (1,0)인 경우는 비 결함 문장, (0,1)의 경우는 실제 결함 문장을 의미한다. 각 시험케이스 별로 이러한 문장별 입력 데이터를 생성 가능하다.

**표 2. 생성된 입력 데이터**

| CI | PF | DU | 결함여부 | |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0.2 | 1 | 0 |
| 1 | 0 | 0.2 | 1 | 0 |
| 1 | 0 | 0.2 | 1 | 0 |
| | | | | |
| … | … | … | … | … |
| 1 | 0.5 | 0.5 | 1 | 0 |

표 2 는 최적화된 인공신경망 모델의 결함 및 비결함 문장 분류 가중치 값을 활용하여 각 문장의 의심도를 산출한 결과를 나타낸다. 해당 경우는 CI 와 PF, DU-쌍의 결함 의심 요소를 모두 포함하고 실제 결함을 가지고 있는 5 번 문장이 0.71 로 가장 높은 의심도를 가지며, 그 이상의 의심도를 가지는 문장이

없기 때문에 1/13 으로 7.7%의 결함지역화 비용을 가지게 된다.

## 4. "NextDate" 사례를 통한 실험

"NextDate" 프로그램은 연, 월, 일을 입력 받아, 다음 날에 해당하는 날짜를 연산하여 출력한다. 프로그램은 28 LOC 이며, 본 실험을 위해 우리는 등가분할(Equivalence Partitioning) 기법과 경계 값 분석(Boundary Value Analysis) 기법을 이용하여 15 개의 시험케이스를 정의하였다. 표 3 은 정의한 시험케이스를 나타낸다.

**표 3. 시험케이스**

| 월 | 일 | 연도 |
|---|---|---|
| 0 | 0 | 1800 |
| 10 | 30 | 2017 |
| 10 | 31 | 2017 |
| 11 | 29 | 2017 |
| 11 | 31 | 2017 |
| 11 | 30 | 2017 |
| 12 | 30 | 2017 |
| 12 | 32 | 2021 |
| 12 | 32 | 2017 |
| 2 | 27 | 2017 |
| 2 | 28 | 2012 |
| 2 | 28 | 2017 |
| 2 | 30 | 2017 |
| 2 | 29 | 2017 |
| 2 | 29 | 2012 |

표 4 는 투입한 서로 다른 유형의 결함인 9 개의 결함 별로 실제 결함 문장의 문장 의심도 순위를 추출하여 생성한 각 결함 별 결함 지역화 비용을 나타낸다. NextDate 코드의 경우에는 3 번, 4 번, 5 번 ,6 번, 9 번 문장의 경우, 논리연산자 오류의 결함에서 실제 결함을 가지고 있는 문장이 높은 의심도를 가진다. 논리연산자 오류의 결함 지역화에 있어 높은 효율을 보여준다.

표 4 의 지역화비용은 28 개 문장을 하나하나 살피며 결함을 찾는 경우에 비해, 추출된 문장 의심도 순위에 따라 결함 지역화를 수행한 경우 실제 결함문장을 발견하는 비용을 나타낸다.

**표 4. 결함유형별 문장의심도 및 비용**

| 결함 유형 | 실제결함 문장의심도 순위 | 지역화비용(%) |
|---|---|---|
| 1 (변수초기화) | 20 | 71.4 |
| 2 (문장누락) | 11 | 39.2 |
| 3 (논리연산자) | 1 | 3.5 |
| 4 (논리연산자) | 5 | 17.8 |
| 5 (논리연산자) | 2 | 7.1 |
| 6 (논리연산자) | 1 | 3.5 |
| 7 (논리연산자) | 25 | 89.2 |
| 8 (변수초기화) | 20 | 71.4 |
| 9 (잘못된논리식) | 4 | 14.2 |

## 5. 결론

본 논문에서는 프로그램 소스코드 문장의 시험케이스의 커버리지 정보, Pass/Fail 여부, DU 정보를 활용하여 각 문장들의 결함 의심도를 산출하는 방법을 제안하고 예제를 통하여 적용하였다. 그 결과, 특정 결함에서는 DU-쌍 가중치가 높은 문장들과 시험케이스가 실패한 경우가 많아 해당 가중치가 높은 문장들에서 높은 결함 의심도가 산출되었고 실제 결함 문장의 의심도를 높게 산정한 것을 확인할 수 있었다. 또한 여러 결함 유형들 중 다른 유형의 결함에 비해 잘못된 논리연산자를 사용한 결함 유형에 있어 사용자가 확인해야 할 문장의 개수가 줄어들었다.

## 참고문헌

[1] W. Eric Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A Survey on Software Fault Localization", IEEE Transactions on Software Engineering, Vol. 42, Issue 8. pp. 707-740, 2016

[2] 문석현, 김윤호, 김문주, "테스트 케이스의 결함 실행 확률을 이용한 결함 위치 추정 기법", Korea Conference on Software Engineering, 2013

[3] R. Santelices, J.A. Jones, Y. Yu, and M.J. Harrold, "Lightweight Fault-Localization using Multiple Coverage Types", International Conference on Software Engineering, 2009

[4] P. Zhang, X. Mao, Y. Lei and Z. Zhang, "Fault Localization Based on Dynamic Slicing via JSlice for Java Programs", International Conference on Software Engineering and Service Science, 2014

[5] A. Perez, R. Abreu, A. van Deursen, "A Test-Suite Diagnosability Metric for Spectrum-based Fault Localization Approaches", International Conference on Software Engineering, 2017

[6] W. Eric Wong and Y. Qi, "BP neural-network based effective fault localization", International Journal of Software Engineering and Knowledge Engineering, Vol. 19, No. 4, pp. 577-593, 2009.