

# 프로젝트에 적응하는 결함 위치 추정 기법

2024. 07. 09

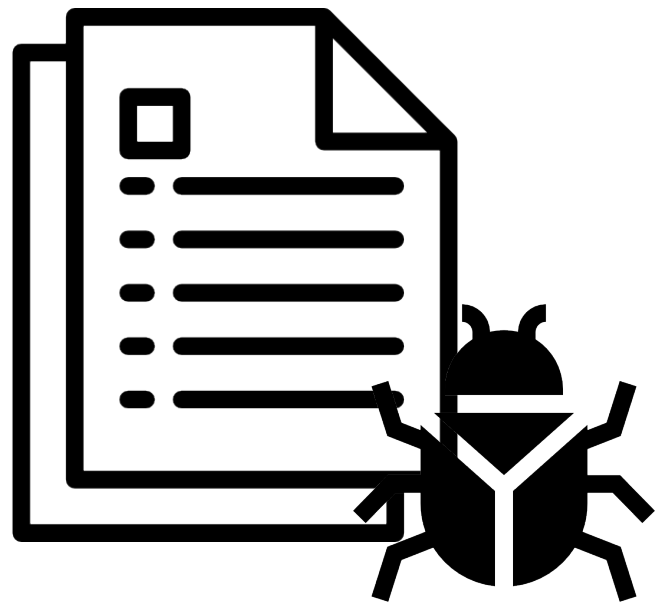
고려대학교 소프트웨어분석연구실

김동욱

# 소프트웨어 오류 디버깅 과정

- 소프트웨어 디버깅은 다음 단계로 구성: 오류 발생 → 위치 추정 → 결함 수정 → 검증
- 빠른 프로그램 수정을 위해 정확한 결함 위치 추정은 디버깅 과정에서 중요한 역할을 함

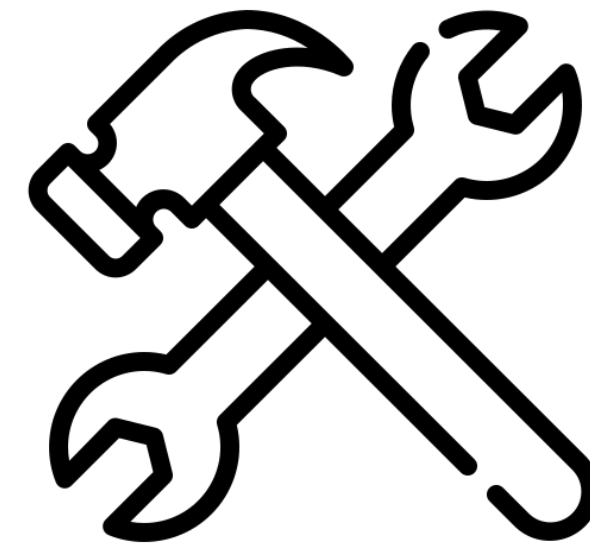
프로그램의 오류 발생



수정할 결함의 위치 추정



결함 수정



수정된 프로그램 검증

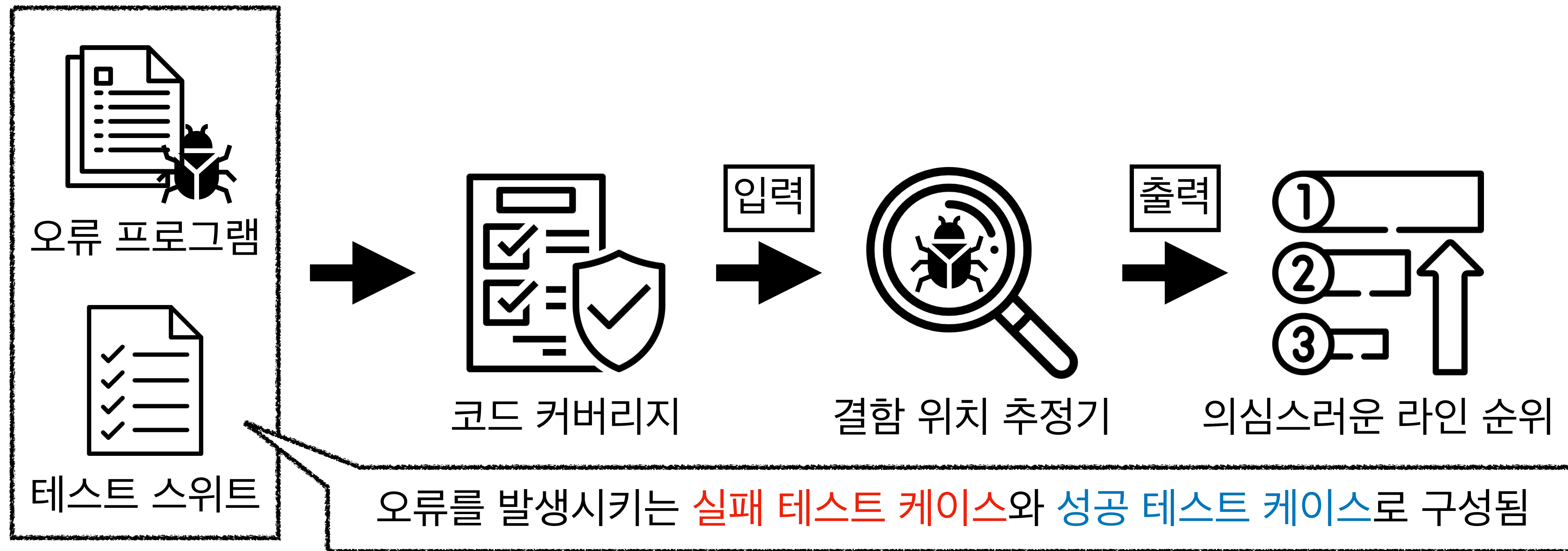


결함 위치 추정의 방법

- Program logging
- Assertions
- Breakpoints
- 자동 결함 위치 추정기

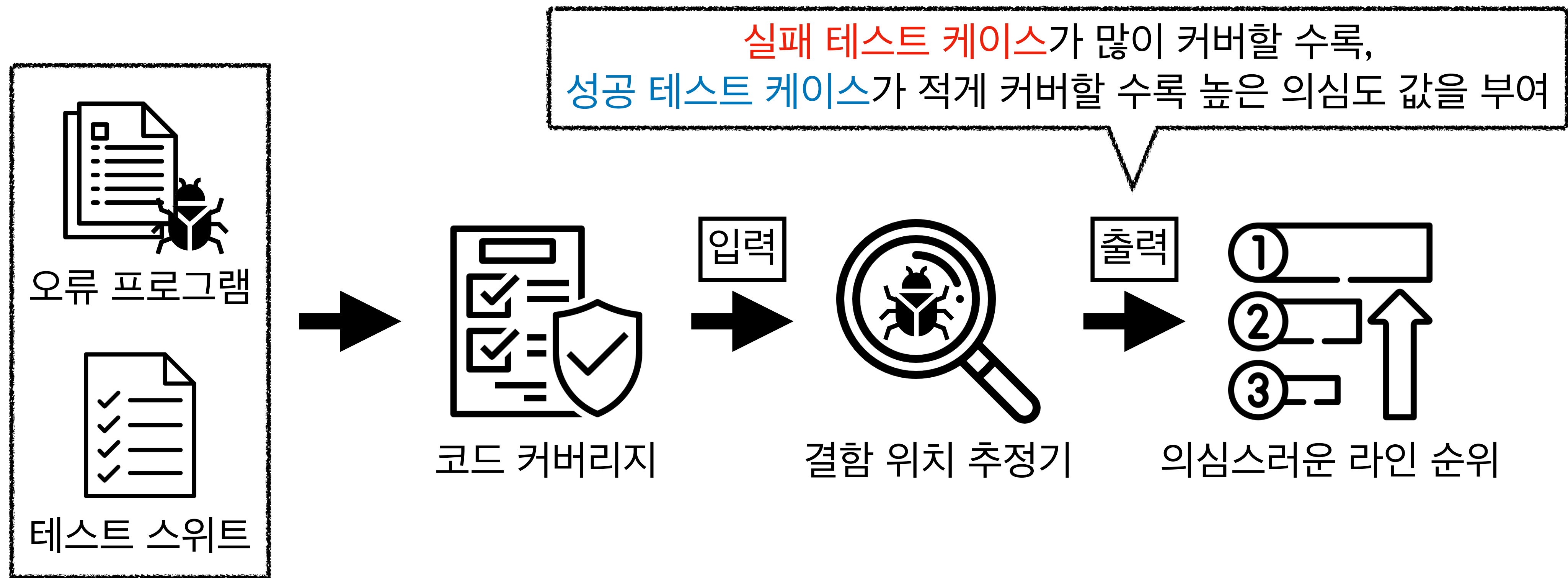
# 결함 위치 추정 기술 (Fault Localization)

- 커버리지 기반의 결함 위치 추정 기술은 코드 커버리지를 입력으로 사용함
- 코드의 라인 별 의심도를 계산하고 의심도 값에 따른 순위를 제공함



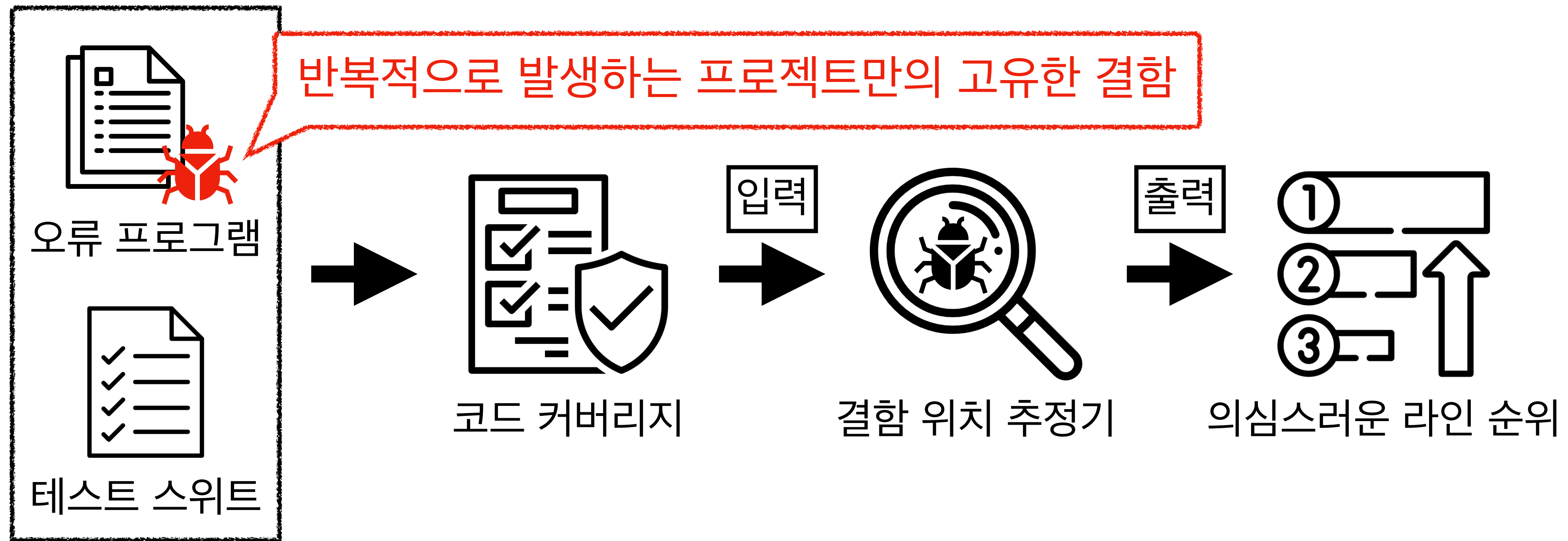
# 결함 위치 추정 기술 (Fault Localization)

- 커버리지 기반의 결함 위치 추정 기술은 코드 커버리지를 입력으로 사용함
- 코드의 라인 별 의심도를 계산하고 의심도 값에 따른 순위를 제공함



# 기존의 결함 위치 추정 기술의 문제점

- 관찰 1: 프로젝트는 목적을 달성하기 위해 특정한 기능을 수행함
  - 관찰 2: 특정한 기능을 수행할 때 주로 발생하는 결함의 패턴이 존재함
- 기존의 FL 기술은 프로젝트 별 결함 패턴을 고려하지 않음



# 관찰: 프로젝트 별 결함 패턴이 존재함

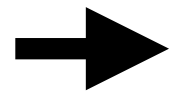
- Cppcheck 프로젝트는 C++ 프로그램을 대상으로 하는 정적 분석기
- 정적 분석을 위해 C++ 소스코드를 파싱해야 하는데, C++ 코드 파싱은 매우 어려운 문제임

→ Cppcheck 프로젝트의 경우, 토큰 패턴 매칭 관련 결함이 지속적으로 발생함

```
bool isTemporary(bool cpp, const Token* tok,...){
    ...
    if (Token::Match(tok->previous(), ...)){
        ...
    }
+ if (tok->isCast())
+   return false;
+ // Currying a function is unknown ...
+ if (Token::simpleMatch(tok, "(") && ...
+   return unknown;
    return true;}

```

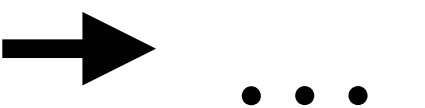
Cppcheck #5



```
static bool isDeadTemporary(bool cpp, ...){
    if (!isTemporary(cpp, tok, library))
        return false;
-   if (expr && !precedes(...))
-       return false;
+   if (expr) {
+       if (!precedes(...))
+           return false;
+       const Token* parent = tok->astParent();
+       // Is in a for loop
+       if (astIsRHS(tok) && ...) {
+           const Token* braces = ...
+           if (precedes(braces, expr) && ...
+               return false;}}
    return true;}

```

Cppcheck #19



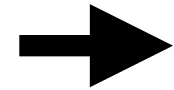
...

# 관찰: 프로젝트 별 결함 패턴이 존재함

- Exiv2 프로젝트는 이미지 메타데이터를 읽기, 쓰기, 삭제, 수정할 수 있는 유틸리티
  - 프로그램의 대부분은 여러 형태의 이미지 메타데이터를 다루기 위한 로직
- Exiv2 프로젝트의 경우, 입력 메타데이터를 검사하는 코드에서 결함이 지속적으로 발생

```
...  
uint32_t resrcLength = getULong(buf, bigEndian);  
+ enforce(resrcLength < io_->size(), ...);  
while (resrcLength > 0)  
...
```

Exiv2 #10



```
...  
subBox.length = getLong(...);  
subBox.type = getLong(..., bigEndian);  
- if (subBox.length > io_->size() - io_->tell()){  
+ if (subBox.length < sizeof(box) || ...) {  
    throw Error(kerCorruptedMetadata);}
```

Exiv2 #13



...

# 관찰: 결함 패턴은 프로젝트마다 다름

- 프로젝트 별로 핵심 결함 패턴이 다름
- Cppcheck: 토큰 패턴 매칭 , Exiv2: 메타데이터 핸들링

```
bool isTemporary(bool cpp, const Token* tok,...){  
    ...  
    if (Token::Match(tok->previous(), ...)){  
        ...}  
+ if (tok->isCast())  
+   return false;  
+ // Currying a function is unknown ...  
+ if (Token::simpleMatch(tok, "(") && ...  
+   return unknown;  
    return true;}
```

Cppcheck #5

VS

```
...  
uint32_t resrcLength = getULong(buf, bigEndian);  
+ enforce(resrcLength < io_->size(), ...);  
while (resrcLength > 0)  
...  

```

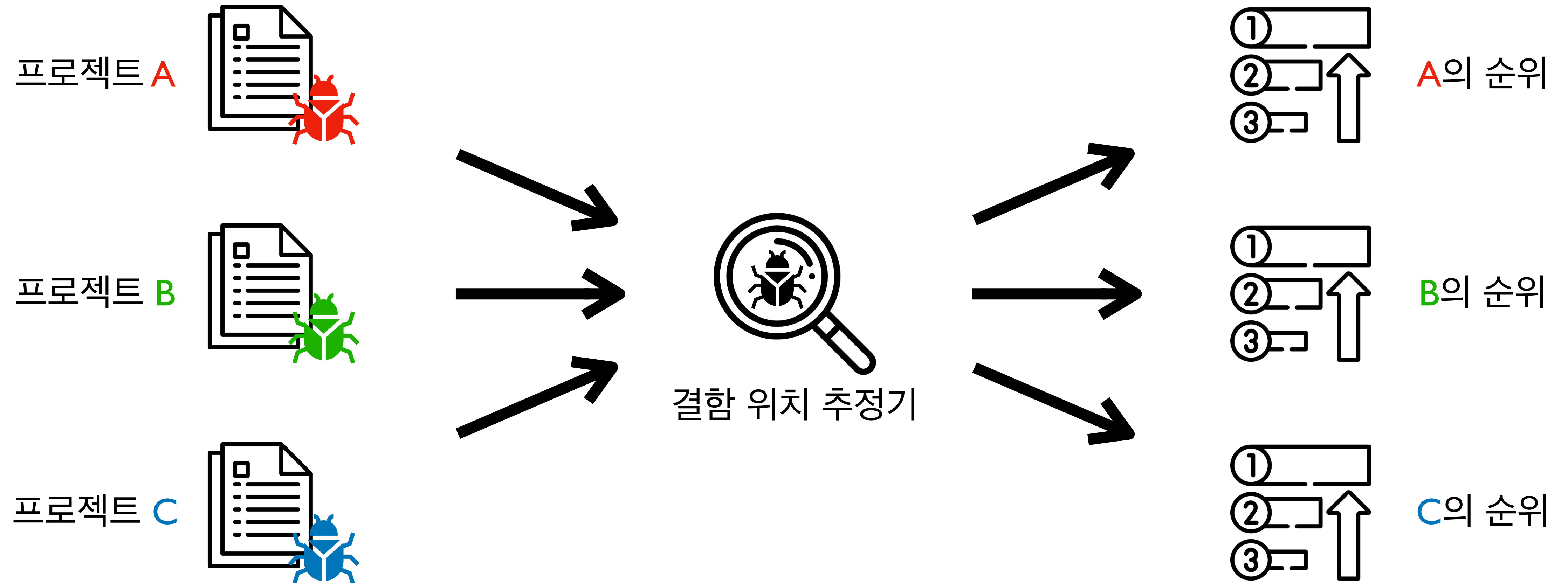
Exiv2 #10

관찰로부터 얻은 직관

프로젝트 별로 특화된 결함 위치 추정기가 필요하다

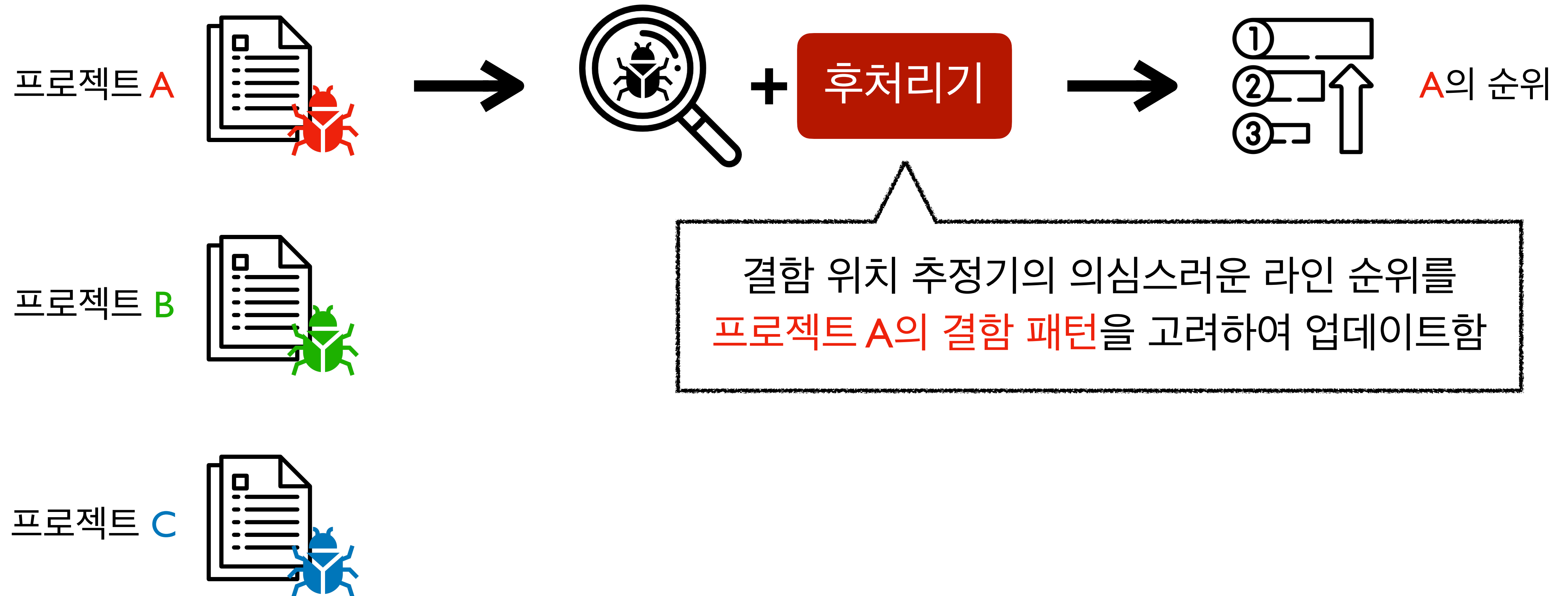
# 기존 기술들의 동작 방식

- 고정된 결함 위치 추정기를 프로젝트 구분 없이 글로벌하게 사용함



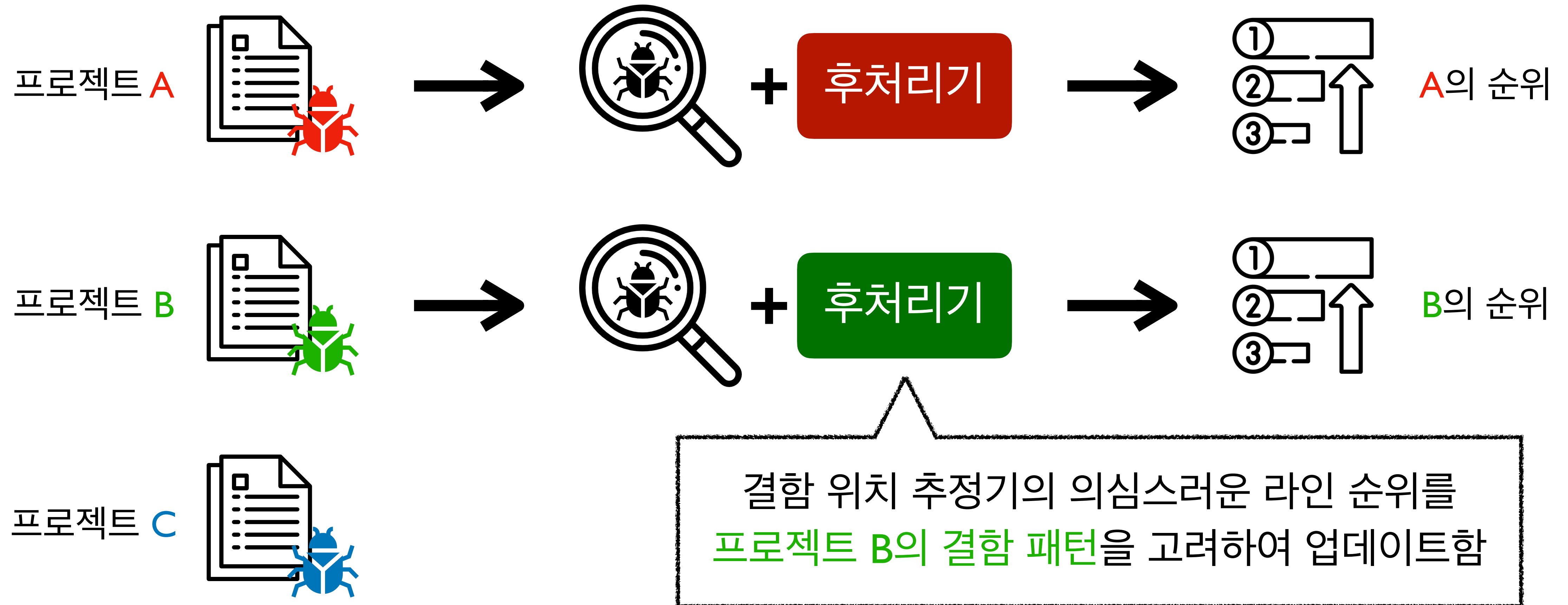
# 프로젝트에 적응하는 동작 방식

- 프로젝트의 결함 패턴을 학습시켜 프로젝트에 특화된 후처리기
- 후처리기는 결함 위치 추정기의 라인 별 의심도를 조정하여 순위를 업데이트함



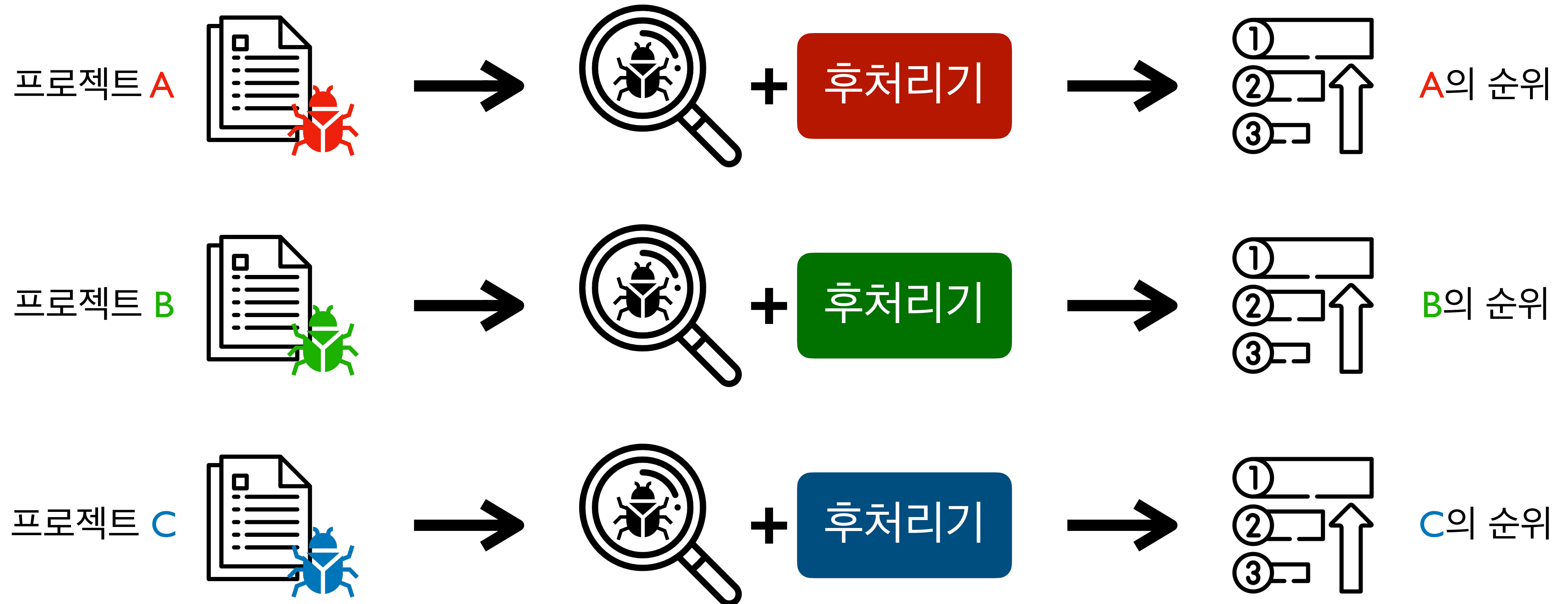
# 프로젝트에 적응하는 동작 방식

- 프로젝트의 결함 패턴을 학습시켜 프로젝트에 특화된 후처리기
- 후처리기는 결함 위치 추정기의 라인 별 의심도를 조정하여 순위를 업데이트함



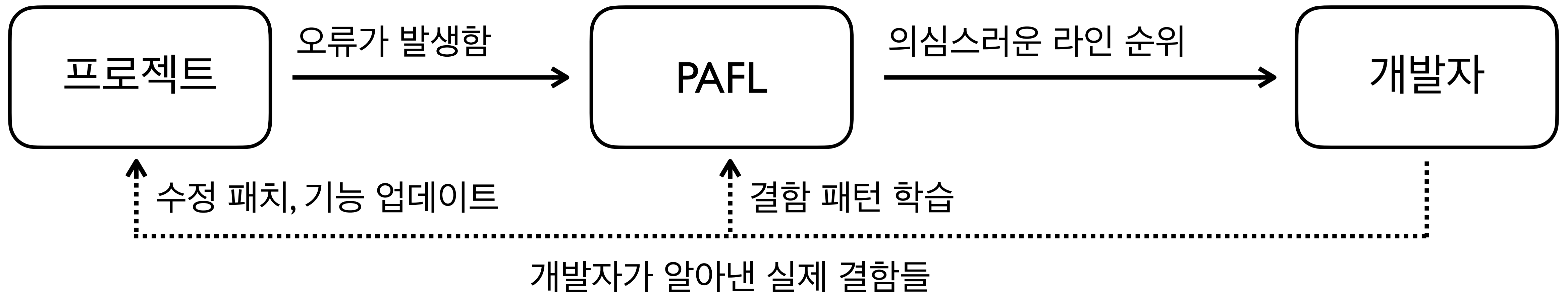
# 프로젝트에 적응하는 동작 방식

- 프로젝트의 결함 패턴을 학습시켜 프로젝트에 특화된 후처리기
- 후처리기는 결함 위치 추정기의 라인 별 의심도를 조정하여 순위를 업데이트함



# PAFL: 프로젝트에 적응하는 결함 위치 추정

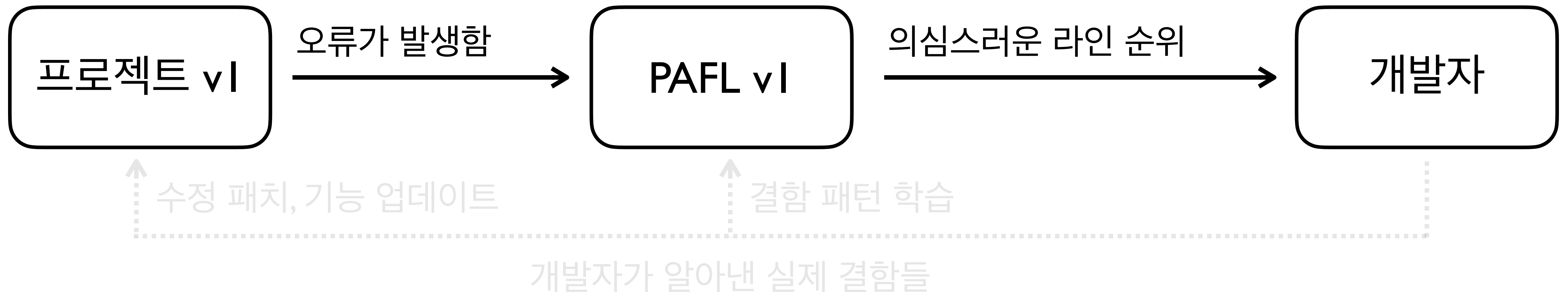
- PAFL은 온라인 러닝을 통해 프로젝트의 결함 패턴을 학습함
- 업데이트 된 PAFL은 다음 버전의 프로젝트의 결함 위치 추정에 사용됨



# PAFL: 프로젝트에 적응하는 결함 위치 추정

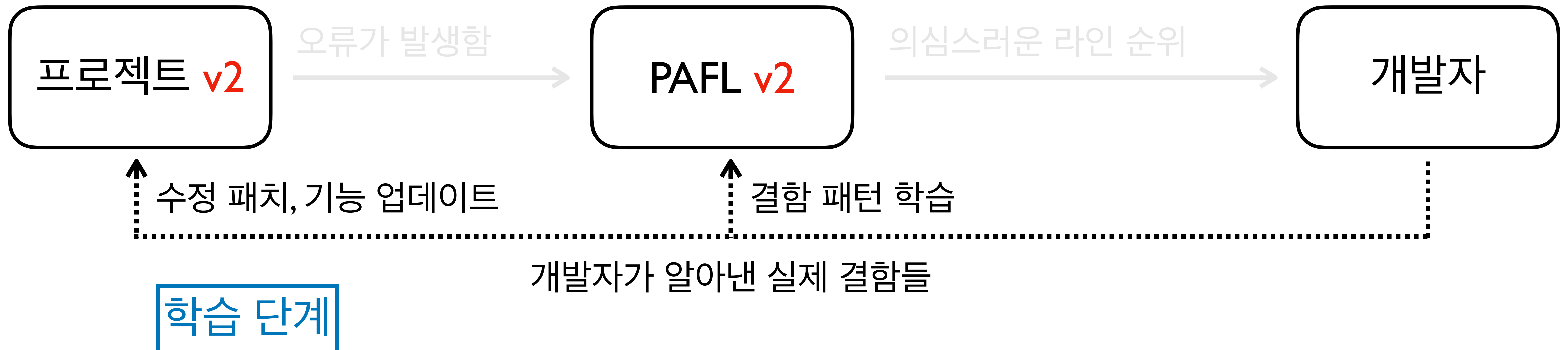
- PAFL은 온라인 러닝을 통해 프로젝트의 결함 패턴을 학습함
- 업데이트 된 PAFL은 다음 버전의 프로젝트의 결함 위치 추정에 사용됨

## 결함 위치 추정 단계



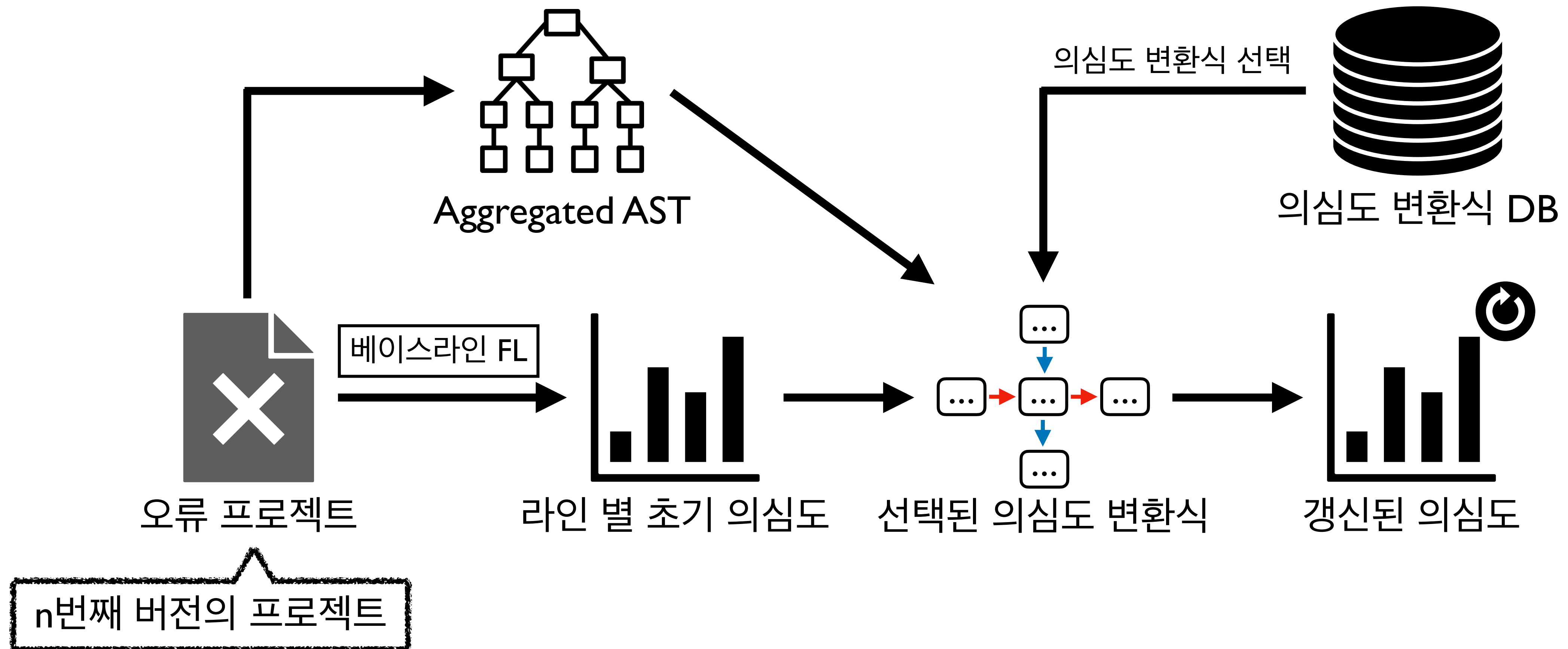
# PAFL: 프로젝트에 적응하는 결함 위치 추정

- PAFL은 온라인 러닝을 통해 프로젝트의 결함 패턴을 학습함
- 업데이트 된 PAFL은 다음 버전의 프로젝트의 결함 위치 추정에 사용됨



# PAFL의 결함 위치 추정 단계

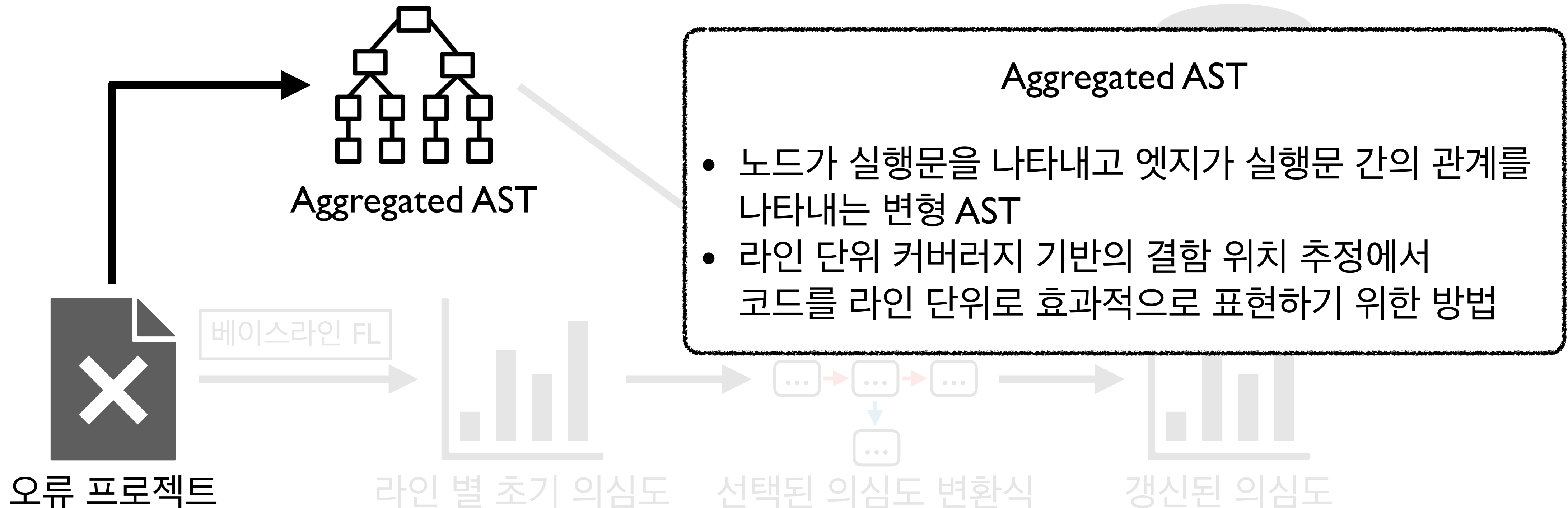
프로젝트의 이전 버전들 (Version 1 ~ n-1)을 데이터로 학습함



# PAFL의 결함 위치 추정 단계

프로젝트의 이전 버전들 (Version 1 ~ n-1)을 데이터로 학습함

## I. 입력 프로그램의 AST에서 Aggregated AST를 생성함



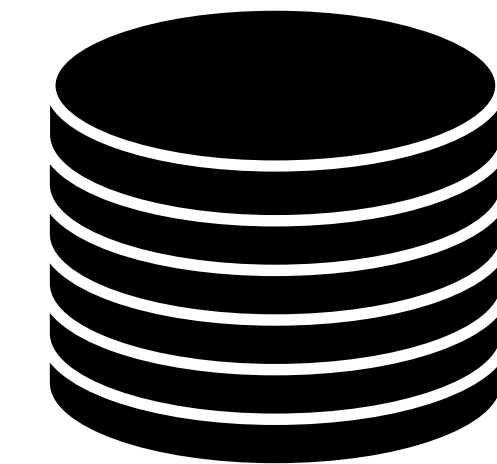
n번째 버전의 프로젝트

# PAFL의 결함 위치 추정 단계

프로젝트의 이전 버전들 (Version 1 ~ n-1)을 데이터로 학습함

- 의심도 변환식은 결함 패턴과 유사한 코드에 의심도를 추가함
- 의심도 변환식을 선택하는 과정은 현재 오류 프로젝트의 결함 패턴을 예측하는 과정

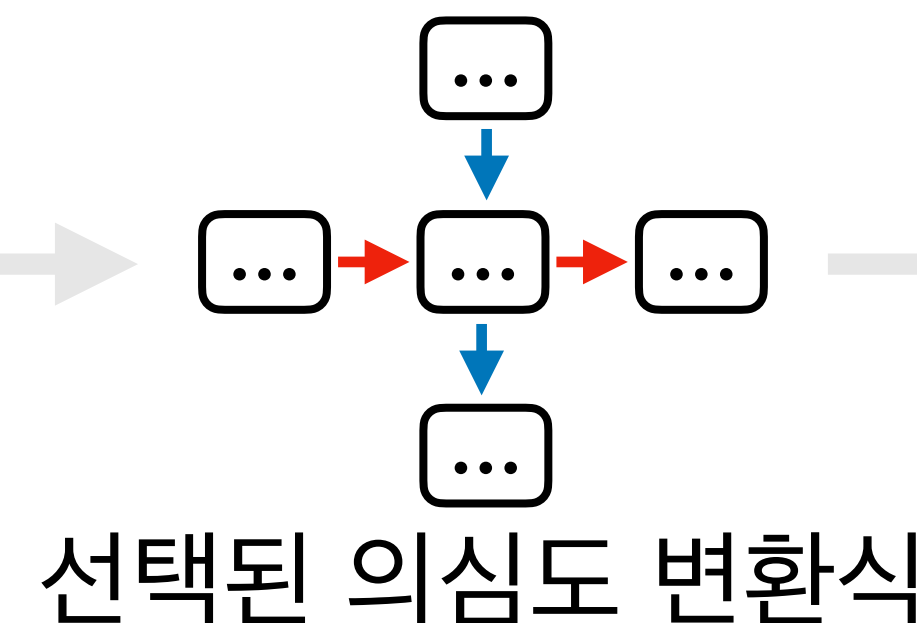
의심도 변환식 선택



의심도 변환식 DB



베이스라인 FL



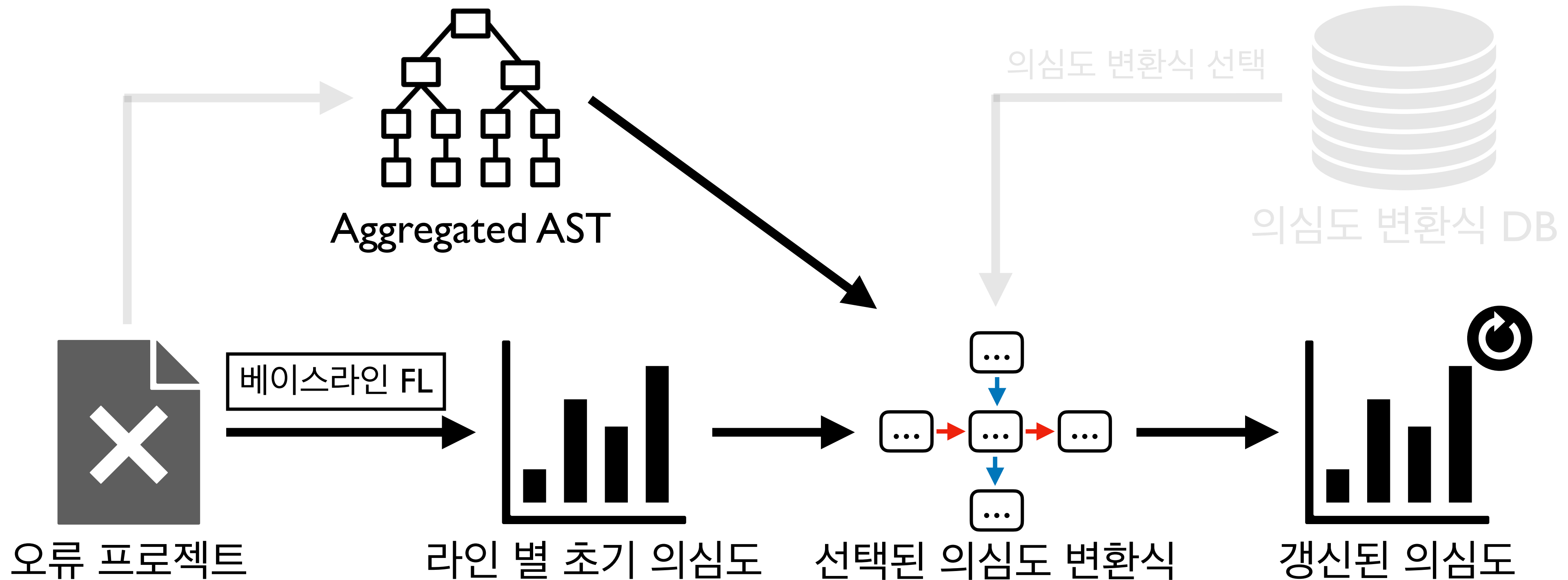
n번째 버전의 프로젝트

2. 의심도를 업데이트할 의심도 변환식을 선택함

# PAFL의 결함 위치 추정 단계

- 의심도 변환식은 타겟 라인이 결함 라인인지 비교하기 위해서 Aggregated AST로 표현된 라인 관계를 참조해 타겟 라인의 주변 라인까지 함께 확인함

학습함

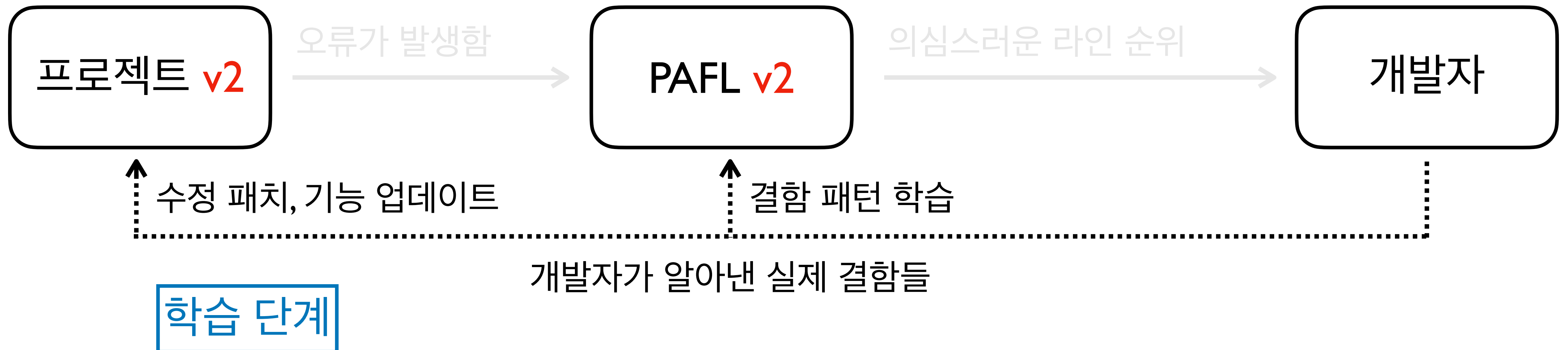


n번째 버전의 프로젝트

3. 의심도 변환식으로 라인 별 의심도를 업데이트함

# PAFL 학습 단계

- PAFL은 온라인 러닝을 통해 프로젝트의 결함 패턴을 학습함
- 업데이트 된 PAFL은 다음 버전의 프로젝트의 결함 위치 추정에 사용됨



# 벤치마크: BugsCpp, BugsInPy

- 총 12개의 프로젝트 (C/C++ 8개, 파이썬 4개)
- 총 224개의 프로젝트의 오류 버전 (C/C++ 139개, 파이썬 85개)
- 6가지 베이스라인 결함 위치 추정기 시나리오

| Language | Project           | # Versions | LoC    | Test LoC | # Tests |
|----------|-------------------|------------|--------|----------|---------|
| C/C++    | <i>cpp-peglib</i> | 10         | 15.4K  | 1.7K     | 134     |
|          | <i>cppcheck</i>   | 30         | 274.9K | 44.6K    | 92      |
|          | <i>exiv2</i>      | 20         | 83.4K  | 17.5K    | 6       |
|          | <i>libchewing</i> | 8          | 106.0K | 3.6K     | 18      |
|          | <i>libxml2</i>    | 7          | 231.0K | 33.5K    | 41      |
|          | <i>proj</i>       | 28         | 215.3K | 42.5K    | 54      |
|          | <i>openssl</i>    | 26         | 947.8K | 102.3K   | 226     |
|          | <i>yaml-cpp</i>   | 10         | 76.8K  | 19.6K    | 15      |
| Python   | <i>thefuck</i>    | 31         | 9.8K   | 6.9K     | 1,649   |
|          | <i>fastapi</i>    | 15         | 23.4K  | 1.7K     | 432     |
|          | <i>spacy</i>      | 7          | 97.2K  | 6.2K     | 1711    |
|          | <i>YouTube-dl</i> | 32         | 119.8K | 15.9K    | 223     |

# Research Question

- (RQ1) **Efficiency of PAFL**
  - PAFL의 베이스라인 결함 위치 추정기의 성능을 얼마나 향상시키는가?
  - PAFL이 일관적으로 성능을 향상시키는가?
- (RQ2) **Comparison with State-of-the-Art**
  - SOTA인 Aeneas (ICSE 2022) 보다 좋은 성능 향상을 보이는가?
  - Aeneas보다 비용이 더 적게 드는가?

실패 테스트 케이스의 커버리지를 합성하는 Oversampling 기술

# RQI - 베이스라인 대비 성능 향상

- 베이스라인을 SBFL로 사용했을 때 실제 결함의 순위가 최소 **14.6% 향상**
- 실제 결함 라인을 1등으로 보고한 경우가 **총합 82.6% 상승**

|         |        | SBFL   |               |        |        |               |       |         |               |       | DLFL       |               |       |        |              |        |             |              |       |
|---------|--------|--------|---------------|--------|--------|---------------|-------|---------|---------------|-------|------------|---------------|-------|--------|--------------|--------|-------------|--------------|-------|
| project | metric | OCHIAI | PAFL          | Delta  | DSTAR  | PAFL          | Delta | BARINEL | PAFL          | Delta | MLP-FL     | PAFL          | Delta | CNN-FL | PAFL         | Delta  | RNN-FL      | PAFL         | Delta |
| total   | MFR    | 2869.0 | <b>2441.4</b> | 14.9%  | 3055.0 | <b>2459.7</b> | 19.5% | 2985.2  | <b>2548.4</b> | 14.6% | 1425.9     | <b>1381.0</b> | 3.1%  | 354.0  | <b>316.5</b> | 10.6%  | 185.6       | <b>184.7</b> | 0.5%  |
|         | MAR    | 3222.3 | <b>2896.6</b> | 10.1%  | 3448.9 | <b>2972.0</b> | 13.8% | 3302.2  | <b>2967.3</b> | 10.1% | 2062.9     | <b>2014.1</b> | 2.4%  | 422.1  | <b>386.8</b> | 8.4%   | 254.9       | <b>253.2</b> | 0.7%  |
|         | Top-1  | 5      | <b>13</b>     | 160.0% | 8      | <b>15</b>     | 87.5% | 8       | <b>15</b>     | 87.5% | <b>6.6</b> | <b>6.6</b>    | 0.0%  | 0.2    | <b>1.2</b>   | 500.0% | <b>0.0</b>  | <b>0.0</b>   | 0.0%  |
|         | Top-5  | 23     | <b>35</b>     | 52.2%  | 25     | <b>38</b>     | 52.0% | 25      | <b>37</b>     | 48.0% | 13.8       | <b>14.4</b>   | 4.3%  | 0.6    | <b>3.0</b>   | 400.0% | <b>6.6</b>  | <b>6.6</b>   | 0.0%  |
|         | Top-10 | 36     | <b>52</b>     | 44.4%  | 36     | <b>51</b>     | 41.7% | 37      | <b>52</b>     | 40.5% | 21.4       | <b>22.8</b>   | 6.5%  | 1.0    | <b>5.4</b>   | 440.0% | <b>13.0</b> | <b>13.0</b>  | 0.0%  |

# RQI - 성능 향상의 일관성

- PAFL이 베이스라인 대비 실제 결함의 순위를 상승시키거나 유지한 비율
- 베이스라인이 Ochiai인 경우, 전체 결함 중 99%가 순위가 하락하지 않음

|                   | OCHIAI | DSTAR | BARINEL | MLP-FL | CNN-FL | RNN-FL |
|-------------------|--------|-------|---------|--------|--------|--------|
| <i>cpp-peglib</i> | 100%   | 100%  | 100%    | 94%    | 92%    | 94%    |
| <i>cppcheck</i>   | 93%    | 87%   | 90%     | 69%    | N/A    | N/A    |
| <i>exiv2</i>      | 100%   | 100%  | 100%    | 87%    | N/A    | N/A    |
| <i>libchewing</i> | 100%   | 100%  | 100%    | 90%    | 72%    | 95%    |
| <i>libxml2</i>    | 100%   | 100%  | 100%    | 83%    | N/A    | N/A    |
| <i>proj</i>       | 100%   | 96%   | 100%    | 69%    | N/A    | N/A    |
| <i>openssl</i>    | 96%    | 92%   | 100%    | N/A    | N/A    | N/A    |
| <i>yaml-cpp</i>   | 100%   | 100%  | 100%    | 90%    | N/A    | N/A    |
| <i>thefuck</i>    | 100%   | 100%  | 97%     | 99%    | 94%    | 95%    |
| <i>fastapi</i>    | 100%   | 100%  | 100%    | 96%    | 93%    | 95%    |
| <i>spacy</i>      | 100%   | 100%  | 100%    | 89%    | 89%    | 89%    |
| <i>YouTube-dl</i> | 100%   | 94%   | 100%    | 93%    | N/A    | N/A    |
| Total             | 99%    | 96%   | 98%     | 86%    | 91%    | 94%    |

최소 86%

# RQ2 - Aeneas와 성능 비교

- 모든 지표에서 베이스라인에 PAFL만을 적용하거나 Aeneas와 동시에 적용한 결과가 가장 좋았음
- Aeneas는 오히려 베이스라인의 성능을 크게 떨어뜨리는 경우가 관찰됨

| method  | scenario | MFR          | MAR           | Top-1     | Top-5     | Top-10    |
|---------|----------|--------------|---------------|-----------|-----------|-----------|
| OCHIAI  | origin   | 749.2        | 933.0         | 5         | 22        | 31        |
|         | AENEAS   | 1926.3       | 2477.0        | 7         | 24        | 33        |
|         | PAFL     | <b>701.6</b> | <b>880.9</b>  | 9         | <b>29</b> | <b>42</b> |
|         | together | 1825.4       | 2168.3        | <b>10</b> | 27        | 36        |
| DSTAR   | origin   | 1005.3       | 1232.8        | 8         | 24        | 31        |
|         | AENEAS   | 3112.6       | 3730.9        | 1         | 6         | 7         |
|         | PAFL     | <b>884.6</b> | <b>1094.6</b> | <b>13</b> | <b>32</b> | <b>41</b> |
|         | together | 2884.5       | 3339.5        | 3         | 8         | 10        |
| BARINEL | origin   | 749.4        | 933.3         | 8         | 24        | 32        |
|         | AENEAS   | 697.2        | 1206.5        | 4         | 19        | 29        |
|         | PAFL     | 697.4        | <b>869.1</b>  | <b>13</b> | <b>31</b> | <b>42</b> |
|         | together | <b>624.8</b> | 945.0         | 10        | 27        | 40        |

# RQ2 - Aeneas와 시간 비용 비교

- Aeneas는 **24시간이 넘게 걸려서** 측정하지 못한 경우도 발생
- 반면, PAFL은 1개의 프로젝트를 제외하면 비용이 **평균 3초 이내**

각 오류 버전 별 평균 오버헤드 (분 단위)

|                   | AENEAS | PAFL |                   | AENEAS | PAFL |
|-------------------|--------|------|-------------------|--------|------|
| <i>cpp-peglib</i> | 0.50   | 0.01 | <i>openssl</i>    | N/A    | 0.22 |
| <i>cppcheck</i>   | 248.26 | 0.03 | <i>yaml-cpp</i>   | 3.02   | 0.01 |
| <i>exiv2</i>      | N/A    | 0.01 | <i>thefuck</i>    | 5.97   | 0.01 |
| <i>libchewing</i> | 2.14   | 0.01 | <i>fastapi</i>    | 1.25   | 0.01 |
| <i>libxml2</i>    | 750.55 | 0.05 | <i>spacy</i>      | 14.7   | 0.03 |
| <i>proj</i>       | N/A    | 0.03 | <i>YouTube-dl</i> | 10.91  | 0.02 |