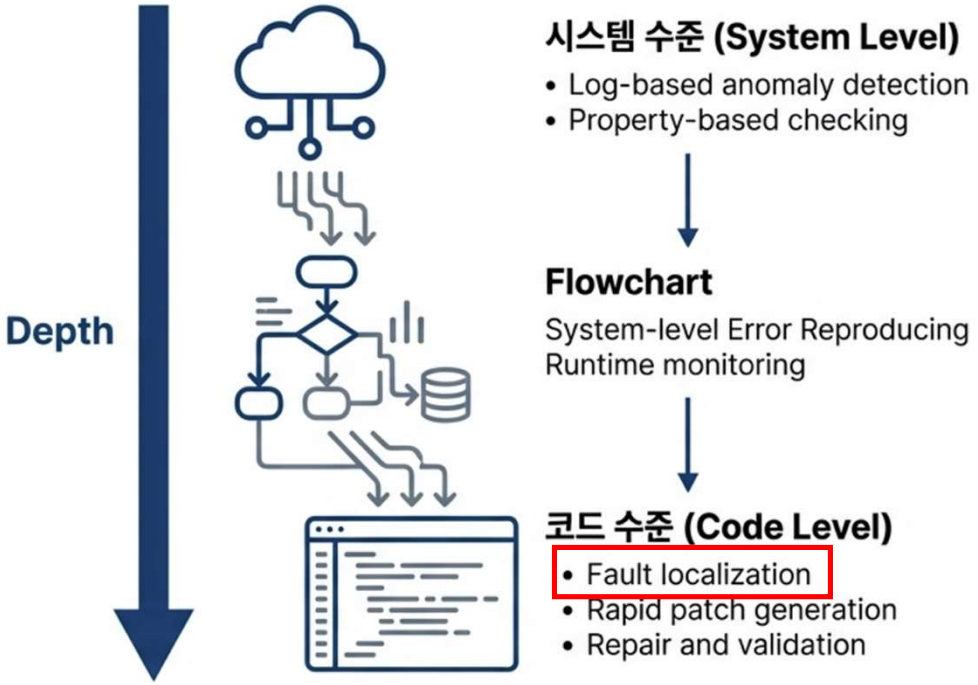
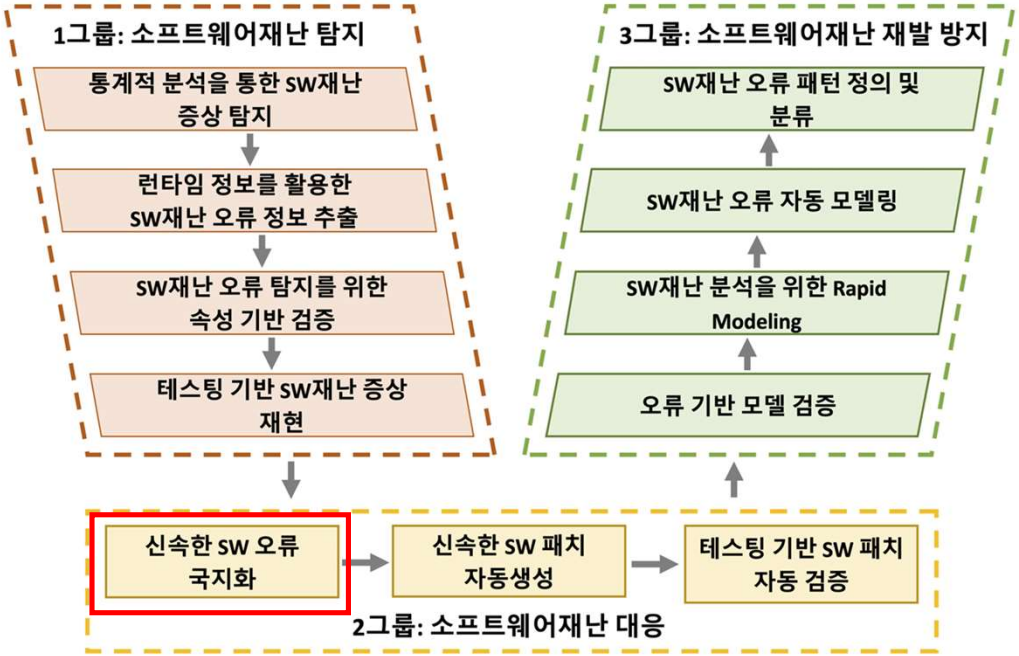
# LLM을 활용한 다단계 결함 위치 탐지

김윤호
한양대학교 소프트웨어공학 연구실

# 센터에서의 역할

- SW재난 신속대응을 위해 오류 위치를 신속, 정확하게 찾기
  - 임무: Kubernetes와 같은 대규모 클라우드 인프라 소프트웨어의 SW 재난 신속 대응 시연



# 기존 LLM 기반 결함 위치 탐지의 한계

- 컨텍스트 제한으로 대규모 프로젝트의 면밀한 단위의 결함 위치 탐지가 어려움
- 기존 LLM 기반 결함 위치 탐지의 선택

### 프로젝트 단위 분석

큰 프로젝트를 듬성듬성 확인

- ✓ 대규모 프로젝트 전체 커버 가능
- ✗ 구체적인 구문(Statement) 위치 특정 불가
- → 결함 **의심 메서드** 제시 수준에 그침

**예시:** SoapFL [Qin et al., TSE 25]

### 모듈 단위 분석

작은 모듈을 면밀하게 확인

- ✓ 결함 **의심 구문**까지 정밀 제시
- ✗ 프로젝트 전체 컨텍스트 반영 불가
- → 사전에 좁혀진 작은 모듈만 분석 가능

**예시:** LLMAO [Yang et al., ICSE 24]

- 임무 달성을 위해 대규모 프로젝트의 결함 위치를 구문단위로 신속, 정확하게 탐지해야 함

# 관찰과 제안

- 관찰: 우리 인류는 지금까지 수 많은 결함을 성공적으로 제거함
  - 논리점프1: 인류의 디버깅 방법론은 적어도 현재까진 제법 훌륭하다
  - 논리점프2: 그렇다면, 인류의 디버깅 방법론은 LLM에게도 적합할 것이다.

- 사람에게 효과적인 디버깅 방법론을 LLM에게 이식해보자
  - 큰 프로젝트에선 각 클래스와 메서드의 의미를 파악하여 의심도를 부여하고
  - 작은 모듈에선 각 구문의 의미를 상세하게 설명하면서 의심가는 구문을 찾아보자

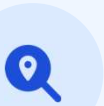
# 사람이 디버깅을 하는 방법

1

## 오류 재현 테스트 생성

오류 상황을 안정적으로 재현하기 위해 최소화된 테스트 케이스 작성

2



## 오류 원인 파악

테스트 실패 정보와 실행 행위를 분석하여 근본적인 결함 위치를 탐지

3

## 오류 수정

파악된 원인을 바탕으로 코드를 수정하여 결함 제거

4

## 검증 및 회귀 테스트

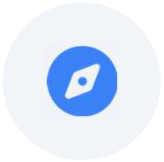
수정이 올바른지 확인하고, 재발 방지를 위한 회귀 테스트 수행

# 사람이 오류 원인을 파악하는 방법

## 결함 이해

테스트 실행 행위와 실패 정보를 분석하여 버그의 가능한 원인(가설)을 도출

- Stack trace 및 로그 분석
- 테스트 커버리지 및 오류 현상 확인



## 코드 탐색

결함 가설을 바탕으로 의심스러운 클래스와 메서드 범위를 점진적으로 좁힘

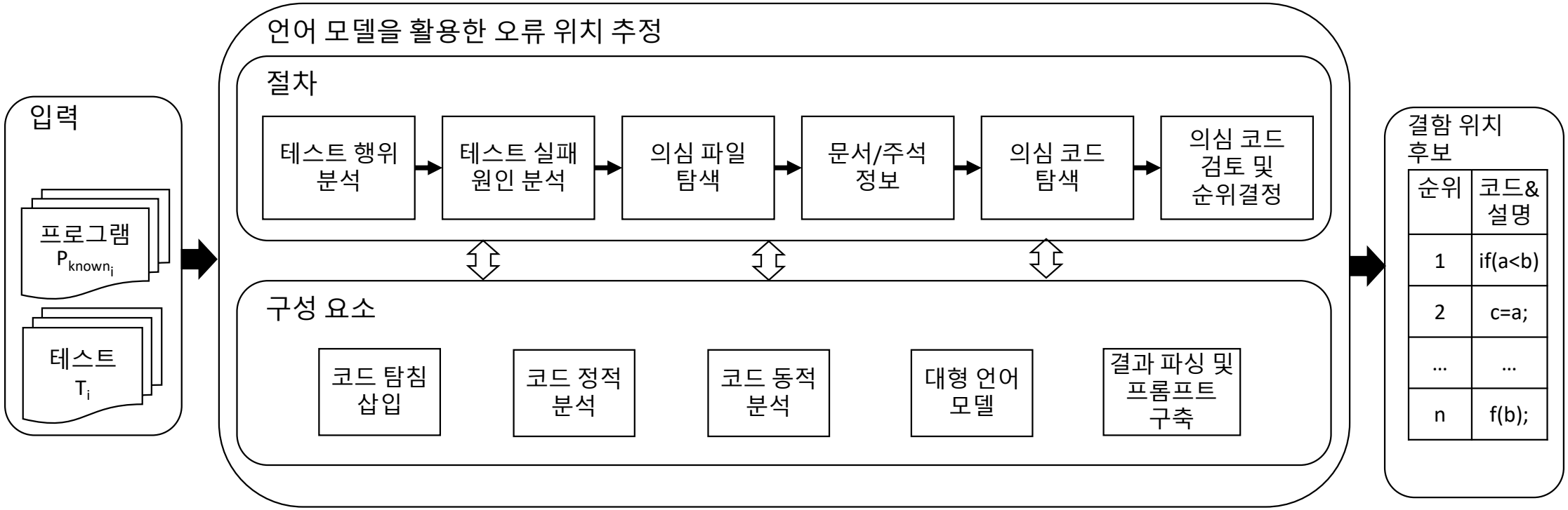
- 후보군 축소 전략
- 관련 파일 및 문서 검토

## 결함 확인

선정된 후보 메서드들을 정밀 검토하여 실제 결함이 포함된 의심 메서드를 확정

- 메서드 상세 리뷰
- 최종 의심 메서드 식별

# LLM을 활용한 다단계 결함 위치 탐지



# LLM을 사용한 결함 이해

- 테스트 실패의 원인(가능한 결함 가설)을 파악하고, 이후 코드 탐색의 가이드라인으로 사용.

- 테스트 행위 분석
  - 실패 테스트가 무엇을 검증하려 했는지를 상세히 설명
  - 단순 테스트 메서드가 아니라, 테스트 유틸리티 메서드까지 포함한 전체 실행 행위를 분석
- 테스트 실패 분석
  - 테스트 행위 + 실패 정보 기반으로 공통된 실패 패턴과 가능한 결함 원인 목록 도출

테스트 클래스 [TEST CLASS]에 포함된
하나 이상의 테스트가 실패했다
실패한 테스트: [FAILED TESTS]

아래에 나열된 실패한 테스트 코드와
관련된 메서드 코드를 보고
[TEST CODE]
당신은 테스트 코드 리뷰어로서,
각 테스트의 코드를 가능한 상세하게
설명해야 한다.

각 테스트를 설명할 때에는, 해당
테스트에서 호출되는 테스트 유틸리티
메서드의 코드 로직도 함께 포함하여
설명하고 필요 시 유틸리티 메서드를
요청하시오.

# LLM을 사용한 결함 이해

- 테스트 실패의 원인(가능한 결함 가설)을 파악하고, 이후 코드 탐색의 가이드라인으로 사용.
- 테스트 행위 분석
  - 실패 테스트가 무엇을 검증하려 했는지를 상세히 설명
  - 단순 테스트 메서드가 아니라, 테스트 유틸리티 메서드까지 포함한 전체 실행 행위를 분석
- 테스트 실패 분석
  - 테스트 행위 + 실패 정보 기반으로 공통된 실패 패턴과 가능한 결함 원인 목록 도출

테스트 클래스 [TEST CLASS]에 포함된
하나 이상의 테스트가 실패했다
실패한 테스트: [FAILED TESTS]

각 실패한 테스트의 테스트 코드, 오류 스택 트레이스, 테스트 출력 결과,
그리고 아래에 나열된 실패 테스트의 동작 특성을 바탕으로
[STACK TRACE]
[TEST RESULTS]
[TEST BEHAVIOR]
당신은 소프트웨어 테스트 엔지니어로서
(1) 주어진 테스트 동작, 출력 결과, 스택 트레이스에서 공통적인 패턴이나 유사점을 식별하고,
(2) 테스트가 실패하게 만들었을 가능성이 있는 코드 상의 결함 후보를 제안해야 한다.

# LLM을 사용한 코드 탐색

- 결함 가설을 바탕으로 프로젝트 전체 코드베이스에서 의심 파일/클래스와 의심 메소드를 단계적으로 좁힘.

- 의심 파일/클래스 탐색
  - 실패 테스트들이 공통으로 거친 클래스 중 가장 의심스러운 클래스 1개 선택
- 의심 메소드 선정
  - 결함과 관련 있을 가능성이 있는 메서드 후보군 선택

테스트 클래스 [TEST CLASS]에 포함된 하나 이상의 테스트가 실패했다
실패한 테스트: [FAILED TESTS]

기존 분석 결과에 따르면 클래스 [CLASS NAME] 에 문제가 있을 가능성이 있으며, 해당 클래스의 문서는 [CLASS DOC] 이다.

각 실패한 테스트의 테스트 코드, 오류 스택 트레이스, 테스트 출력 결과, 테스트 실패의 가능 원인, 그리고 아래에 나열된 해당 클래스의 메서드 목록을 바탕으로
[TEST INFOS]
[POSSIBLE CAUSES]
[COVERED METHODS]
당신은 소프트웨어 엔지니어로서 Covered Methods 목록을 검토하여, 테스트 실패의 원인이 되었을 가능성이 있는 모든 메서드를 선택해야 한다.
단, 반드시 Covered Methods 목록에 포함된 메서드만 선택해야 한다.

# LLM을 사용한 결함 확인

- 앞 단계에서 모은 후보 메서드들을 정밀 검토하여 최종 의심도 순위를 산출.

- 메서드 리뷰
  - 후보 메서드를 하나씩 LLM이 검토
  - 테스트 실패와의 인과관계를 바탕으로 의심도 점수(1~100) + 설명 생성
- 의심 메서드 순위
  - 점수 기준으로 메서드 정렬

- 여러 메서드를 한 번에 주지 않고 메서드당 1회 추론으로 집중도 유지

테스트 클래스 [TEST CLASS]에 포함된 하나 이상의 테스트가 실패했다
실패한 테스트: [FAILED TESTS]

기존 분석 결과에 따르면 메서드 [METHOD NAME] 에 문제가 있을 가능성이 있다.

각 실패한 테스트의 테스트 코드, 오류 스택 트레이스, 테스트 출력 결과, 테스트 실패의 가능 원인, 그리고 아래에 제시된 의심 메서드(suspicious method) 정보를 바탕으로
[TEST INFOS], [POSSIBLE CAUSES]
[CLASS NAME], [CLASS DOC]
[METHOD NAME], [METHOD DOC]
[METHOD CODE]
당신은 소프트웨어 엔지니어로서 메서드 [METHOD NAME] 의 코드를 **한 줄씩 면밀히 검토**하여, 이 메서드가 결함 원인일 가능성이 얼마나 높은지 평가해야 한다.

응답은 반드시 #SCORE #DESCRIPTION 형식으로 작성해야 하며, 여기서 SCORE는 1~100 사이의 정수 점수로 해당 메서드가 얼마나 의심스러운지를 나타내고, DESCRIPTION은 그 점수를 준 이유를 자연어로 설명한 것 이다.

# 의심 메서드에서 의심 구문 찾기

- 러버덕 디버깅
  - 오리 인형에게 말로 코드를 설명하다 보면 문제가 되는 구문을 파악하게 되는 놀라운 디버깅 기법
  - “자 오리야 이 메서드 첫 줄은 지역 변수를 선언한거고 둘째 줄은 첫 번째 파라메터가 정상적인지 체크하는거야. 정상적인 파라메터의 기준은 ….. 스물 세번째 줄은 … 어 왜 여기서 함수 리턴값이 널인지 체크가 빠졌지?”

- 왜 효과적인가?
  - 시각 외에 청각이라는 감각을 추가적으로 활용하여 사람의 인지 능력을 높임
  - 암묵적인 구문 이해를 명시적인 설명으로 전환하는 과정에서 놓치고 있던 문제 발견

- LLM에게도 러버덕 디버깅이 효과적일 것
  - 각 구문을 설명하게 함으로써 분석의 컨텍스트를 좁혀 면밀하게 살펴보며 기존 메서드 단위 분석에서 놓칠 수 있는 오류를 효과적으로 탐색

# 향후 연구

- 구체적인 테크닉 완성 및 구현

- 실험 평가: 그래서 정말 결함 위치 탐지를 잘 하는가?
  - Defects4J의 Chart의 2~3개 결함에 적용해본 결과 기존에 결함 위치 탐지가 잘 안되던 결함에서 TOP 5 수준의 좋은 결과 보임
  - Kubernetes와 같은 실전 대규모 프로젝트에서의 유용성 입증 필요