대규모 언어 모델을 활용한 단위 테스트의 적합성 자동 식별

김대원, 이영규, 유준범
건국대학교 컴퓨터공학부
kdwkdw0078@gmail.com, sontouf@gmail.com, jbyoo@konkuk.ac.kr

Automated Identification of Unit Test Adequacy Using Large Language Models

Daewon Kim, Younggyu Lee, Junbeom Yoo
Department of Computer Science and Engineering, Konkuk University

요 약

소프트웨어가 수정됨에 따라 단위 코드의 동작이 변경되면, 기존 단위 테스트가 여전히 의도된 동작을 검증하고 있는지 판단하기 어렵다. 단위 테스트의 적합성 여부를 확인하기 위해 개발자가 코드를 직접 분석해야 하므로 많은 시간과 노력이 필요하다. 본 논문에서는 대규모 언어 모델(LLM)을 활용하여 단위 테스트의 적합성 여부를 자동으로 식별하는 기법을 제안한다. 제안 기법은 수정 전후의 소스 코드와 테스트 코드를 입력으로 받아, 선행 연구에서 정의한 변경 유형 분류 체계와 판단 기준을 프롬프트에 구조화하여 LLM 이 테스트의 검증 대상 유지 여부를 판단하고 구체적인 판단 근거를 제공하도록 설계되었다. GoogleTest 샘플을 대상으로 한 사례 연구 결과, LLM 은 함수 대체나 재귀 변환과 같이 명확한 변경 유형에서는 테스트 적합성을 효과적으로 식별하였으나, 복잡한 경우에는 일관된 판단에 어려움이 있었다. 본 연구는 LLM 을 활용한 단위 테스트 적합성 자동 식별의 가능성과 한계를 함께 확인하고, 이를 개선하기 위한 향후 연구 방향을 제시한다.

1. 서 론

단위 테스트는 소프트웨어를 구성하는 개별 기능이 명세에 따라 올바르게 동작하는지를 검증하는 핵심 수단이다. 그러나 소프트웨어가 지속적으로 수정되는 상황에서 단위 코드와 테스트 코드 간의 공진화(co-evolution)가 적절히 이루어지지 않으면, 테스트가 수정된 코드의 동작을 더 이상 검증하지 못하는 상황이 발생할 수 있다[1, 2].

선행 연구에서는 테스트 실행 시 발생하는 함수 호출과 객체 간 상호작용을 동적으로 분석하여 시퀀스 다이어그램으로 시각화하고, 테스트 변화 유형을 체계적으로 분류하였다. 이를 통해 개발자는 코드 수정 전후의 실행 흐름 차이를 시각적으로 파악하고, 테스트가 여전히 의도된 동작을 검증하는지 판단할 수 있는 기준을 확보하였다. 하지만, 시퀀스 다이어그램의 변화가 테스트 검증 대상의 실질적 변경인지, 단순한 리팩토링인지를 판단하는 과정은 여전히 개발자의 수작업에 의존한다는 한계가 있다[3, 4].

본 논문에서는 대규모 언어 모델(Large Language Model, LLM)을 활용하여 단위 테스트의 적합성 여부를 자동으로 식별하는 기법을 제안한다. 제안 기법은 수정 전후의 소스 코드와 테스트 코드를 입력으로 받아, 선행 연구의 16 개 변경 유형 분류 체계와 판단 기준을 프롬프트에 구조화하여 LLM 이 테스트의 검증 대상 유지 여부를 판단하도록 한다. 동적 실행이나 계측 없이 정적 코드 분석만으로 작동하며, 판단 결과와 함께 구체적인 근거를 제공하여 개발자의 의사결정을 지원한다[4].

본 논문의 구성은 다음과 같다. 2 장에서는 테스트 적합성 식별 기준과 LLM 기반 자동 분류 기법을 제안하며, 3 장에서는 GoogleTest 샘플을 대상으로 한 사례 연구를 통해 제안 기법을 평가한다. 4 장에서는 연구의 한계와 향후 연구 방향을 논의하고, 5 장에서 결론을 맺는다.

2. LLM 을 활용한 단위 테스트 적합성 자동 식별

단위 테스트의 적합성은 테스트가 의도한 검증 대상과 실제 검증하는 대상의 일치 여부로 판단된다. 개발자가 테스트를 작성할 때는 특정 함수 또는 객체의 특정 동작을 검증하려는 명확한 의도가 존재한다. 선행 연구는 코드 수정 후에도 통과하지만 더 이상 의도된 동작을 검증하지 못하는 테스트를 NLT(No-Longer-Testable)로 정의하였다[3]. 한편, 최근에는 대규모 언어 모델(LLM)을 활용하여 테스트 생성, 테스트 오라클 지원, 테스트 유지보수 등 다양한 소프트웨어 테스트 작업을 자동화하려는 연구가 활발히 진행되고 있다[5, 6]. 본 연구에서는 단위 테스트의 적합성 여부를 LLM 을 활용하여 자동으로 식별하기 위한 판단 기준을 다음과 같이 제시한다.

테스트 적합성 판단에는 세 가지 요소가 고려되어야 한다.

① 테스트가 실제로 호출하는 함수나 메서드가 수정 전후에 동일한지 확인해야 한다. 함수 이름이 같더라도 오버로딩, 오버라이딩, 조건문 변경 등으로 다른 구현이 실행될 수 있다.

② 호출하는 함수가 동일하더라도 그 함수의 입출력 관계가 변경되었는지 확인해야 한다. 내부 구현 방식만 바뀌고 결과가 동일하다면 테스트는 여전히 유효하지만, 같은 입력에 다른 출력을 생성한다면 테스트는 부적합하다.

③ 코드 변경이 명세 갱신에 따른 것인지 리팩토링에 따른 것인지 구분해야 한다. 함수 시그니처 변경이나 메서드 제거는 명세 변경을 시사하며, 변수명 변경이나 내부 로직 재구성은 리팩토링을 시사한다.

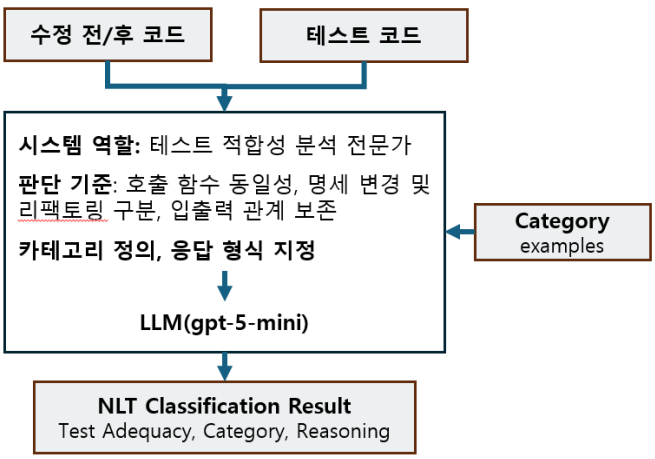

그림 1. 단위 테스트 적합성 자동 식별 과정

<그림 1>은 제안 기법의 전체 과정을 보여준다. 제안 기법은 수정 전 소스 코드, 수정 후 소스 코드, 테스트 코드를 입력으로 받는다. 프롬프트 생성기는 이들 코드와 선행 연구의 16 개 변경 유형 카테고리를 결합하여 구조화된 프롬프트를 생성한다. LLM 은 테스트 적합성 여부, 해당 카테고리, 판단 근거를 출력한다.

프롬프트는 다음 네 부분으로 구성된다. 시스템 역할에서는 LLM 을 테스트 적합성 분석 전문가로 설정하고 테스트 의도와 실제 검증 대상 비교에 집중하도록 지시한다. 판단 기준에는 2 절의 세 가지 요소를 명시하여 LLM 이 함수 동일성, 입출력 관계, 명세 변경 여부를 판단하도록 유도한다. 변경 유형은 <표 1>과 같이 부적합 유형 8 개와 적합 유형 8 개로 제공한다. 각 유형에 특성과 예시를 포함하여 LLM 이 상황을 카테고리에 매칭하도록 한다. 입력 코드는 수정 전후 소스 코드와 테스트 코드를 명확히 구분하여 제시한다.

LLM 의 출력은 분류 결과(적합/부적합), 변경 유형, 판단 근거로 구조화된다. 프롬프트는 증거 기반 추론을 명시적으로 요구하여, LLM 이 어떤 함수가 어떻게 변경되었고 변경 후에도 검증이 유지되는지를 구체적으로 서술하도록 한다. 이를 통해 개발자는 판단 근거를 검토하고 테스트 수정 방향을 결정할 수 있다.

표 1. 단위 테스트 적합성 변경 유형 카테고리

Type	Category	Description
NLT	Overloaded Call	테스트가 다른 오버로드된 메서드를 호출
	Overridden Call	기존 호출에서 오버라이드된 메서드로 변경
	Binding Change	가상 바인딩으로 인해 호출 대상 변경
	Hierarchy Shift	상속 계층 구조의 변경
	Condition Change	조건 로직으로 인해 호출 대상 변경
	Output Change	동일 입력에 대해 다른 출력
	Type Mismatch	입력 데이터 타입 변경
	Call Replacement	다른 동작을 가진 메서드 호출 대체
Non-NLT	Variable Renaming	동일 동작의 변수명 변경
	Method Renaming	동일 동작의 메서드명 변경
	Helper Replacement	동등한 헬퍼 메서드로 대체
	Instance Switch	다른 인스턴스에 의한 동일 동작 실행
	Delegated Call	다른 객체로 호출 위임
	Recursion Replacement	반복문을 재귀 호출로 대체
	Order Change	호출 순서 변경
	Mediator Adjustment	중간 객체 변경으로 동작 영향 없음

3. 사례연구

본 사례 연구는 변경 유형별 영향을 LLM 이 식별하는지 확인하기 위해 GoogleTest 에서 제공하는 10 개 샘플 중 변경 유형을 명확히 적용할 수 있는 8 개 샘플만을 선정하였다[7]. 나머지 2 개 샘플은 테스트 프레임워크 동작을 다루는 예제로, 본 논문에서 정의한 코드 변경 유형을 적용하기에 적합하지 않아 사례 연구 대상에서 제외하였다. 각 샘플에 대해 선행 연구에서 정의한 16 개 변경 유형 카테고리를 모두 적용하여 총 128 개의 테스트 케이스를 생성하였다. LLM 으로는 GPT-4 mini 를 사용하였으며, 2 장에서 제시한 프롬프트 구조를 적용하였다. 각 테스트 케이스에 대해 LLM 이 출력한 식별 결과(적합/부적합)와 카테고리를 정답과 비교하여 정확도를 측정하였다. 카테고리는 코드 수정의 의도와 테스트의 원래 검증 대상을 기준으로 선행 연구에서 사전에 정의되었다[3]. <그림 2>는 시스템 프롬프트의 일부분이고, <그림 3>은 프롬프트에 사용되는 카테고리 형식, <그림 4>는 LLM 의 응답 예시이다.

```
def get_system_prompt() -> str:
    return """You are an expert software testing analyst specializing in test validity assessment.
Your expertise includes:
- Understanding test intent and specification alignment
- Analyzing code changes and their semantic impact
- Evaluating runtime execution behavior changes
- Distinguishing between implementation changes and behavioral changes

Your task is to classify unit tests as either "NLT (No-Longer-Testable)" or
"Non-NLT" based on whether they still validate their intended behavior after code modifications.

CRITICAL RULES:
1. Focus on whether the TEST still validates its ORIGINAL INTENT
2. Consider both code-level semantics and execution-level behavior
3. Distinguish between "different implementation" and "different behavior"
4. Always provide clear, evidence-based reasoning
5. Use only the predefined category names exactly as given
6. Base your analysis on comparing before and after states
7. When multiple files are involved, consider their interactions and dependencies

OUTPUT FORMAT (strict):
Classification: [exactly "NLT" or "Non-NLT"]
Category: [exact category name from provided list]
Reasoning: [clear explanation with specific evidence]"""
```

그림 2. 테스트 적합성 식별을 위한 시스템 프롬프트

```
NON_NLT_CATEGORIES = {
    "Variable Renaming": {
        "description": "Variable names changed but the behavior and logic remain identical.",
        "example": "int count renamed to int total, same computation. ...
    }, ...
```

그림 3. 프롬프트에 사용되는 카테고리 형식

```
{  "classification": "Non-NLT",
   "category": "Recursion Replacement",
   "reasoning": "The original intent of the test was to validate the functionality of
   the Factorial function for negative, zero, and positive inputs. ...
}
```

그림 4. LLM 응답 결과

<표 2>는 128 개 테스트 케이스에 대한 LLM 의 분류 결과를 정답과 비교한 것이다. 총 128 개 케이스 중 44 개를 완전히 정확하게 분류하여 34.4%의 전체 정확도를 기록하였다. <그림 5>는 128 개 케이스의 분류 결과 분포를 시각화한 것으로, 일치 44 개(34%), 카테고리 불일치 36 개(28%), 불일치 39 개(31%), 카테고리 오류 9 개(7%)로 나타났다.

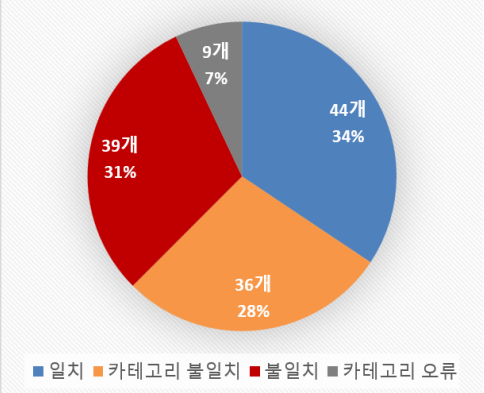

그림 5. 테스트 적합성 예측 결과

표 2 LLM 예측 결과

Type	Category	Accuracy
NLT	Overloaded Call	50%
	Overridden Call	50%
	Binding Change	25%
	Hierarchy Shift	37.5%
	Condition Change	0%
	Output Change	50%
	Type Mismatch	37.5%
	Call Replacement	50%
Non-NLT	Variable Renaming	12.5%
	Method Renaming	12.5%
	Helper Replacement	25%
	Instance Switch	37.5%
	Delegated Call	50%
	Recursion Replacement	50%
	Order Change	25%
	Mediator Adjustment	37.5%

LLM 의 응답 결과는 크게 4 가지로 분류되었다. 먼저 카테고리만 오분류한 경우이다. 36 개 케이스에서 LLM 은 NLT/Non-NLT 적합성은 정확히 판단했으나 세부 카테고리를 잘못 분류하였다. 이 중 17 개는 NLT 유형 내에서 카테고리만 혼동한 경우이며, 19 개는 Non-NLT 유형 내에서 카테고리만 혼동한 경우이다. LLM 은 함수 호출 대상이 변경되는 여러 NLT 카테고리들 (Overloaded Call, Type Mismatch, Hierarchy Shift 등) 간의 미묘한 차이를 구분하는 데 어려움을 보였으며, 구현 변경 방식이 유사한 Non-NLT 카테고리들(Instance Switch, Mediator Adjustment, Delegated Call 등) 간에서도 혼동이 발생하였다. 또한 제시된 카테고리내에서 분류하지 못했지만 LLM 자체적으로

Specification Alignment 등 카테고리를 생성하여 Non-NLT 로 성공적으로 식별하였다. 두번째로 적합성 자체를 오판한 경우이다. 39 개 케이스에서 LLM 은 NLT 를 Non-NLT 로 잘못 판단하였다. 주목할 점은 모든 적합성 오판이 NLT→Non-NLT 방향으로만 발생했으며, Non-NLT→NLT 로 오판한 경우는 보이지 않았다. 이는 LLM 이 코드 변경의 심각성을 과소평가하는 경향이 있음을 보인다. 입출력 관계의 실질적 변화, 가상 함수 바인딩 변경, 오버로딩으로 인한 다른 함수 호출 등 실제로는 테스트의 검증 대상이 바뀐 상황을 단순한 구현 변경으로 잘못 인식하였다. 세번째로 판단 자체를 하지 못한 경우이다. 9 개 케이스에서 LLM 이 제시된 16 개 카테고리 내에서 분류하지 못하고 'Specification Alignment', 'No Significant Change', 'Behavioral Equivalence' 등의 자체 카테고리를 생성하였다. 주로 LLM 이 조건문 변경에 따른 실행 경로의 차이를 인식하지 못한 경우로, 코드 수준 분석의 한계를 보여준다. 마지막으로 정확히 분류된 44 개 케이스는 모두 명확한 패턴적 특징을 공유하였다. 반복문과 재귀 호출 간의 구조적 전환, 변수명이나 메서드명만 변경된 경우, 동등한 헬퍼 함수로의 대체, 명시적인 위임 패턴 등 코드 수준에서 변경의 의미를 직관적으로 파악할 수 있는 경우에 LLM 의 정확도가 높았다.

해당 사례 연구는 몇 가지 한계를 갖는다. 첫째, GoogleTest 샘플만을 대상으로 하여 실험 규모가 제한적이며, 실제 산업 프로젝트의 복잡한 테스트 케이스에 대한 검증이 필요하다. 둘째 34.4%의 정확도는 실용적 활용에 제약이 있다. 특히 적합성을 오판하는 30.5%의 경우(39 개 케이스)는 개발자에게 잘못된 신뢰를 주거나 불필요한 작업을 유발할 수 있어 위험하다. 셋째, 현재는 정적 코드만을 입력으로 사용하였으나, 선행 연구의 실행 로그나 시퀀스 다이어그램 정보를 추가로 활용하면 정확도를 개선할 수 있을 것이다. 넷째, 카테고리 간 혼동 유형을 분석한 결과 유사한 의미의 카테고리에서 빈번한 오류가 발생하였으므로, 프롬프트 최적화를 통해 이러한 카테고리 간 판단 기준을 더욱 구체화할 필요가 있다.

4. 결론

본 논문은 LLM 을 활용하여 단위 테스트의 적합성을 자동으로 식별하기 위한 기법을 제안하였다. 선행 연구의 16 개 변경 유형 카테고리와 판단 기준을 프롬프트에 구조화하여, 테스트 적합성을 판단하도록 설계하였다. GoogleTest 샘플을 활용한 사례 연구에서 LLM 은 명확한 변경 유형에서는 효과적으로 작동하였으나, 유사 카테고리 간 구분과 복잡한 경우에는 한계가 있음을 확인하였다.

향후 연구는 다음 방향으로 진행할 계획이다. 첫째, 프롬프트 최적화를 통해 유사 카테고리의 판단 기준을 더욱 구체화하고 명확하게 예시를 제공한다. 둘째, 단위 테스트를 직접 실행하여 실행 로그와 프롬프트를 같이 LLM 의 입력으로 활용하여 실행 수준의 변화를 더 정확히 파악하도록 한다. 셋째, 대규모 실제 프로젝트를 대상으로 검증하여 실용성을 평가한다. .

Acknowledgment

본 연구는 원자력안전위원회의 재원으로 소형모듈원자로규제연구추진단의 지원을 받아 수행한 원자력 안전연구사업의 연구결과입니다. (No. 1500-1501-409)

참고문헌

[1] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, “The oracle problem in software testing: A survey,” IEEE Transactions on Software Engineering, vol. 41, no. 5, pp. 507–525, 2015.

[2] 김대원, 이영규, 허윤아, 유준범, "단위 테스트코드 간 추적성 복구를 위한 시나리오 기반 시각화 도구의 비교 및 필요성 연구" 한국정보과학회 학술발표논문집, 2025. pp. 376-378

[3] Y. Lee, D. Kim, and J. Yoo, “How can we find out that unit tests no longer test the unit?” in 19th IEEE International Conference on Software Testing, Verification and Validation (ICST), submitted

[4] B. Cornelissen, A. van Deursen, L. Moonen, and A. Zaidman, “Visualizing testsuites to aid in software understanding,” in 11th European Conference on Software Maintenance and Reengineering (CSMR’07), 2007, pp. 213–222.

[5] S. Rahman, S. Kuhar, B. Cirisci, P. Garg, S. Wang, X. Ma, A. Deoras, and B. Ray, “UTFix: Change Aware Unit Test Repairing using LLM,” Proc. ACM Program. Lang., vol. 9, no. OOPSLA1, art. 85, pp. 1–26, Apr. 2025

[6] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, “Software Testing With Large Language Models: Survey, Landscape, and Vision,” IEEE Transactions on Software Engineering, vol. 50, no. 4, pp. 911–936, Apr. 2024,

[7] GoogleTest Team, “GoogleTest samples,” GoogleTest Documentation, [Online]. Available: https://google.github.io/googletest/ [last accessed Jan. 13, 2026.]