Python 입문자를 위한 TDD 환경에서 AI 기반 테스트 케이스의 교육 효과 분석

(Evaluating the Educational Impact of AI-Generated Test Cases in a Test-Driven Development Environment for Python Beginners)

손새봄*, 송지영**

(Saebom Son, Jiyoung Song)

요 약

최근 소프트웨어 개발에서는 테스트 자동화와 TDD(Test-Driven Development)가 코드 품질 확보와 학습 효과 향상의 핵심 전략으로 주목받고 있으며, GPT와 같은 인공지능을 활용한 테스트 케이스 자동 생성이 교육 현장에 도입되고 있다. 그러나 단순한 테스트 수의 확대만으로는 충분하지 않으며, 입문자를 대상으로 한 교육적 효과에 대한 실증적 근거도 부족하다. 본 연구는 Python 입문자를 대상으로 TDD 기반 실습 환경을 마련하여, 연구자 설계, GPT 자동 생성, GPT+전략 지시, 학생 작성 테스트 케이스를 비교 분석하고, 테스트 구성 전략의 질이 오류 검출과 교육적 효과에 미치는 영향을 검증하였다. 실험 결과, 연구자 설계 원칙을 반영한 GPT+전략 지시 케이스가 연구자 설계보다 더 높은 오류 검출 성능과 효율을 보였다. TDD 기반 실습은 단순한 코드 검증을 넘어 학습자의 사고력과 오류 인지 역량을 강화하는 데 기여하였다. 이러한 결과는 프로그래밍 교육과정 설계, AI 기반 학습 도구 개발, 그리고 소프트웨어 품질 보증 교육 현장에 활용될 수 있을 것이다.

■ 중심어 : 테스트 주도 개발 ; AI 테스트 케이스 ; 입문자 프로그래밍 교육

Abstract

Test automation and Test-Driven Development (TDD) have emerged as key strategies for improving code quality and learning effectiveness, while artificial intelligence tools such as GPT are increasingly applied to automated test case generation. However, simply increasing the number of test cases or relying on automation tools does not guarantee sufficient fault detection performance, and empirical evidence of their educational effectiveness for novices remains limited. This study implements a TDD-based practice environment for Python beginners and compares researcher-designed cases, GPT-generated cases, GPT with strategy-guided prompts (GPT+Strategy), and student-written cases. Results show that GPT+Strategy cases achieved higher fault detection and efficiency than even the researcher-designed cases. TDD-based practice strengthened learners' logical thinking and error recognition skills. These findings provide practical implications for programming curriculum design, AI-assisted learning tool development, and software quality assurance education.

■ keywords : Test-Driven Development ; AI-generated test cases ; beginners' programming education

Ⅰ. 서 론

소프트웨어 개발에서 테스트 자동화는 코드 품질을 확보하기 위한 핵심 전략 중 하나이다. 개발 초기 단계에서 오류를 발견하면 이를 수정하는 데 드는 비용을 크게 줄일 수 있기 때문에, 많은 프로젝트에서는 단위 테스트와 같은 자동화된 테스트 기법이 활발히 활용되고 있다. 테스트 주도 개발(Test-Driven Development, TDD)은 테스트 코드를 먼저 작성한 뒤, 이를 통과하도록 기능을 구현하는 방식으로, 코드의 견고성과 오류 예방 효과를 높이는 효과적인 접근법이다. TDD는 산업계뿐만 아니라 소프트웨어 교육 현장에서도 점점 더 많은 관심을 받고 있다. 최근에는 인공지능(AI)의 발달로 GPT 언어 모델 등

* 준회원, 한남대학교 컴퓨터공학과
** 종신회원, 한남대학교 컴퓨터공학과

접수일자 : 2025년 08월 29일
게재확정일 : 2025년 09월 23일
교신저자 : 송지영 e-mail : jysong@hnu.kr

다양한 도구를 활용한 테스트 케이스 자동 생성 기술이 등장하면서, 입문자나 학습자가 TDD 환경을 쉽게 경험할 가능성이 확대되고 있다.

AI의 활용은 Python 입문자에게 학습 부담을 줄이고 실습 장벽을 낮추는 데 도움이 될 수 있다. 그러나 테스트 케이스의 단순한 양적 증가나 자동화 도구의 활용만으로는 테스트 품질이 향상된다고 보기 어렵다. 입문자를 대상으로 한 교육적 효과에 대한 실증적 근거는 아직 충분하지 않다. 따라서 TDD 환경에서 AI가 생성한 테스트 케이스의 실행 효율성과 교육적 효과, 그리고 활용 가능성에 관해 분석하는 연구가 필요하다.

본 연구는 Python 입문자를 대상으로, TDD 기반 실습 환경에서 테스트 케이스 구성 전략의 질이 오류 검출 효율성과 교육적 효과에 미치는 영향을 실험적으로 분석한다. 우리는 삼각형 판별 프로그램을 실험 모델로 설정하고, 직접 작성한 테스트 케이스, ChatGPT가 자동 생성한 테스트 케이스, 그리고 학생이 작성한 테스트 케이스의 성능을 비교하였다. 연구자가 직접 설계한 전략을 GPT 프롬프트에 반영하여 생성한 GPT+전략 지시 케이스를 별도로 구성함으로써, 전략적 설계와 AI 자동화가 결합 될 때 성능과 교육적 효과가 어떻게 달라지는지도 검증하였다.

본 논문의 구성은 다음과 같다. 2장에서는 TDD 및 AI 테스트 생성과 관련된 선행 연구를 검토하고, 3장에서는 실험 설계 방법 및 분석 결과를 제시한다. 4장에서는 결과의 교육적 시사점을 논의하며, 5장에서는 결론 및 향후 연구 방향을 제시한다.

Ⅱ. 관련 연구

1. 테스트 주도 개발 (TDD)

TDD는 단순한 개발 기법을 넘어 테스트 우선 사고와 지속적인 리팩토링을 통해 소프트웨어의 품질을 점진적으로 개선하는 철학으로 자리 잡았다[1]. TDD는 코드 결함 감소와 유지보수성 향상에 효과적인 방법으로 제시되어 왔으며, 개발 생산성, 테스트 코드 품질, 테스트 통과율 측면에서 높은 성과를 보였다는 실험 결과도 보고되었다[2]. 이러한 특성으로 인해 TDD는 산업계뿐만 아니라 교육 현장에서도 큰 관심을 받아왔으며, 활발히 활용되고 있다. 소프트웨어 품질 향상은 물론 학습자의 논리적 사고력과 코드 이해 능력 향상에도 긍정적 영향을 미친다는 연구 결과가 보고되었다[2,3]. 기존 연구들은 TDD의 품질·교육 효과를 다수 보고했으나, 정성적 평가에 머무르거나 입문자 대상의 정량적 비교가 부족하다는 한계가 있다.

이를 보완하기 위해 본 연구는 입문자 TDD 실습 환경에서 테스트 케이스의 질적 구성과 오류 탐지 성능을 정량화하고, 학습자 작성 테스트와 AI 생성 테스트를 동일 과제·동일 실행 조건에서 비교하여 교육적 효과를 함께 평가한다.

2. 테스트 케이스 수와 테스트 품질

테스트 케이스의 수가 많다고 해서 오류 탐지 능력이 비례하여 향상되는 것은 아니라는 연구 결과도 보고되었다. 단순히 테스트 수를 늘리는 것보다, 오류 유발 가능성이 큰 경곗값과 비정상 입력을 포함하는 전략적 테스트가 더 효과적이라는 분석 결과가 제시되었으며[4], 테스트 커버리지가 높을수록 오류 검출률이 향상된다는 연구 결과도 함께 보고되었다[5]. 따라서 단순한 테스트 수보다, 전략적 구성과 커버리지 확보가 오류 탐지 성능에 더 큰 영향을 미친다는 점이 강조된다. 선행 연구는 테스트 수량보다 전략·커버리지의 중요성을 제시했지만, 본 연구는 입문자를 대상으로 테스트 개수와 구성 전략의 차이가 성능에 미치는 영향을 실험적으로 보인다.

3. 인공지능 기반 테스트 케이스 생성

최근 GPT 계열 대규모 언어 모델 인공지능(AI)을 활용한 테스트 케이스 자동 생성에 관한 연구가 활발히 진행되고 있다. AI 모델이 코드 분석과 테스트 케이스 작성에서 일정 수준 이상

의 정확도를 보였다는 연구 결과가 보고되었으며[6], GPT 모델로 생성된 테스트 케이스의 오류 탐지 효과를 실험적으로 검증한 연구도 있다[7]. 이러한 연구들은 테스트 설계의 자동화를 통해 개발 비용 절감 및 생산성 향상에 실질적으로 기여할 수 있음을 보여준다. 그러나 생성된 테스트 케이스의 질적 구성을 체계적으로 평가하지 않거나, 입문자 교육 맥락에서 정량 지표로 효과를 검증하지 못했다. 특히 입문자 대상의 프로그래밍 교육에서, AI 생성 테스트 케이스가 학습자의 코드 이해력 및 오류 인지 능력에 어떤 영향을 미치는지 실증적으로 분석한 연구는 아직 초기 단계에 머물러 있다.

본 연구는 이러한 공백을 보완하기 위해 GPT의 단순 생성 케이스와 함께 연구자 전략을 반영한 GPT+전략 지시 케이스를 구성하여 비교 분석한다.

Ⅲ. 본 론

1. 실험 설계 및 환경

본 연구는 Python 입문자 수준에 적합한 삼각형 판별 프로그래밍을 실험 대상으로 선정하였다. 이 문제는 세 변의 길이를 입력받아 정삼각형, 이등변 삼각형, 부등변 삼각형, 또는 삼각형이 성립하지 않는 경우를 구분하는 프로그램으로, 입력 조건과 경계 사례가 명확하여 테스트 설계 실습에 적합하다.

실험에 사용된 코드는 총 35개이며, Python을 배운 지 1년 미만의 입문자인 대학생들이 실제로 설계한 삼각형 판별 함수이다. 이 중 일부는 정답 코드였고, 나머지는 경곗값 누락, 논리 오류, 조건 분기 오류 등 다양한 오류를 포함하고 있어 테스트 성능 평가용 데이터 세트로 활용되었다. 오류 예제에는 세 변의 길이 조건을 잘못 판단하거나, 정삼각형과 이등변 삼각형의 조건을 혼동하는 논리적 오류가 포함된 코드들이 존재한다.

각 함수는 독립된 Python 모듈로 구성되었으며, 테스트 자동화에는 Python 환경의 Pytest 프레임워크를 사용하였다. 각 테스트 케이스에 대해 예상 결과(expected result)를 사전에 정의한 뒤, 실행 결과(actual result)와 비교하여 일치 여부를 판별하였고, 일치하지 않는 경우는 오류 검출로 기록하였다. 반대로 정답으로 올바르게 판별된 경우는 테스트 통과로 구분하였다. 모든 테스트는 동일한 Pytest 설정과 코드 베이스 하에서 반복 적용되었으며, 오류 데이터는 일관된 방식으로 수집하여 실험 조건을 통제하고, 테스트 결과의 신뢰도를 확보하였다.

실험 평가 지표로는 테스트 결함 검출 효율(Faults per Test)을 사용하였다. 테스트 결함 검출 효율은 총 오류 수를 테스트 케이스 수로 나눈 값으로, 하나의 테스트 케이스가 평균적으로 유도한 오류의 수를 나타낸다. 이 지표는 테스트 케이스 수가 서로 다른 그룹 간에도 비교가 가능하도록 고안되었다.

2. 테스트 케이스 구성 전략

테스트 케이스의 작성 주체와 개수에 따라 세 가지 그룹으로 실험을 구성하였다. 첫 번째 그룹은 본 연구자가 설계한 테스트 케이스로, 정삼각형, 이등변 삼각형, 부등변 삼각형, 무효 입력(예: 음수, 0), 삼각형 성립 조건 위반 등 다양한 입력 시나리오를 고려하여 총 7개의 테스트 케이스로 구성되었다. 이 그룹은 실험의 기준점 역할로 다른 그룹(예: AI 생성 테스트, 학생 작성 테스트)의 성능을 평가하는 비교 기준으로 활용되었다.

7개의 테스트 케이스는 다음의 다섯 가지 테스트 설계 전략을 기반으로 구성되었다.

첫째, 등가 분할 기법을 적용하여, 유효 입력(정삼각형, 이등변 삼각형, 부등변 삼각형)과 무효 입력(삼각형이 성립하지 않음, 음수, 0 등)을 구분하고, 각 등가 클래스에서 대푯값을 선정하였다. 둘째, 경곗값 분석 전략을 사용하여 삼각형 성립 조건의 임계점에 해당하는 입력값(예: (1, 2, 3), (10, 1, 1))을 포함시킴으로써, 로직의 경계

처리 능력을 검증하였다. 셋째, 비정상 입력 테스트 전략에 따라 음수나 0과 같은 비정상 입력을 포함하여 잘못된 입력에 대한 프로그램의 견고성을 평가하였다. 넷째, 모든 테스트 케이스는 사전에 정의한 정답(예상 출력)과 실행 결과를 비교하는 오라클 기반 테스트 방식으로 자동 평가되며, 테스트 실패 여부를 객관적으로 판별하였다. 다섯째, 조건 분기 테스트 관점에서, 삼각형 분류 로직의 모든 조건문에 대해 최소 하나 이상의 입력이 포함되도록 설계하여, 코드의 주요 분기 경로를 모두 커버하였다.

두 번째 그룹은 ChatGPT를 활용해 자동으로 생성된 테스트 케이스이며, 개수에 따라 각각 3개(GPT-3), 5개(GPT-5), 7개(GPT-7)로 구성되었다. 동일한 문제 설명을 기반으로 AI에게 테스트 생성을 요청했으며, 생성된 테스트는 수정 없이 그대로 실험에 사용되었다. 테스트 케이스 생성에는 ChatGPT 무료 버전의 기본 모델인 GPT-4o를 사용하였다. 이 그룹은 테스트 수의 변화에 따른 성능 차이뿐만 아니라, AI 생성 테스트의 구성 전략을 분석하는 목적도 있다.

GPT 기반 테스트 케이스는 연구자가 다음과 같은 간단한 프롬프트를 입력하여 생성하였다: “삼각형 판별 프로그램의 테스트 케이스 3개를 생성해줘.”, “삼각형 판별 프로그램의 테스트 케이스 5개를 생성해줘.”, “삼각형 판별 프로그램의 테스트 케이스 7개를 생성해줘.”.

GPT에게 연구자 설계 전략을 명시적으로 지시하는 프롬프트를 제시하여 별도의 테스트 케이스 집합을 구성하였다(GPT+전략 지시). 이 그룹은 연구자 케이스와 동등한 조건에서 비교하기 위해 총 7개의 테스트 케이스로 구성되었다. 이 프롬프트는 등가 분할, 경곗값 분석, 비정상 입력, 조건 분기 커버리지, 오라클 기반 평가 등 연구자가 활용한 전략을 반영하도록 작성되었으며, Pytest 환경에서 실행가능한 형태로 요구되었다. GPT+전략 지시 프롬프트는 다음과 같다.

“삼각형 판별 프로그램을 위한 테스트 케이스 7개를 Python으로 작성해줘. 세 변의 길이를 통해 정삼각형, 이등변 삼각형, 부등변 삼각형, 삼각형 성립 불가를 확인할 수 있어야 해. 다음의 조건을 반드시 반영해서 작성해줘.

① 등가 분할: 각 삼각형 유형과 무효 입력을 대표하는 값 포함

② 경곗값 분석: (1, 2, 3), (10, 1, 1) 등 삼각형 성립 경계 포함

③ 비정상 입력: 음수, 0과 같은 잘못된 입력

④ 조건 분기 커버리지: 모든 분기(정삼각형, 이등변 삼각형, 부등변 삼각형, 성립 불가)를 최소 1회 이상 포함

⑤ 오라클 기반: 예상 출력과 실제 결과를 비교하는 assert 문으로 작성”

세 번째 그룹은 학생들이 자신의 삼각형 판별 프로그램을 검증하기 위해 직접 작성한 테스트 케이스로 구성되었다. 이 그룹은 대부분 정상 입력 위주의 단순한 테스트로 이루어져 있으며, 경계 조건이나 예외 처리와 같은 복잡한 입력 사례의 비중은 상대적으로 낮았다. 일부 학생이 테스트 코드를 작성하지 않은 경우, 해당 사례는 Pytest 실행 대상에서 제외하도록 설정하였고, 해당 데이터는 분석에서 제외하였다.

모든 테스트 케이스 그룹은 동일하게 Python의 Pytest 자동 실행이 적용되었으며, 입력 데이터는 일관된 형식으로 구성하여, 그룹 간 공정한 비교가 가능하도록 실험 조건을 통일하였다.

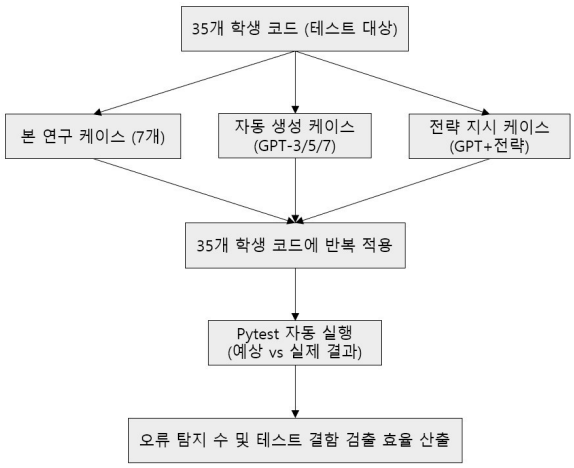

그림 1. 실험 설계 개요

표 1. 테스트 케이스 그룹별 판별 결과

항목	세부 항목	GPT-3 (3개)	GPT-5 (5개)	GPT-7 (7개)	GPT+전략 지시 (7개)	본 연구 설계 (7개)	학생 작성 케이스
테스트 통과 유형별 분포	정삼각형	28	28	28	28	28	18
	이등변 삼각형	29	29	57	29	29	22
	부등변 삼각형	0	31	31	31	31	21
	무효	13	27	62	66	72	14
소계	테스트 통과	70	115	178	154	160	62
	오류 검출	35	60	67	91	85	13
	총 테스트 실행 수	105	175	245	245	245	75

본 실험 설계의 개요는 그림 1에서 보이듯이, 연구자 설계 테스트 케이스, GPT 기반 테스트 케이스 집합, 그리고 학생들이 직접 작성한 테스트 케이스 집합을 35개 학생 코드에 반복 적용한다. 이어서 Pytest로 각 실행의 예상값과 실제값을 자동으로 비교하고 평가하며, 그 결과로 오류 탐지 수와 테스트 결함 검출 효율을 산출한다.

3. 실험 결과

실험 결과, GPT-3, GPT-5, GPT-7은 테스트 개수 증가에 따라 오류 검출 수는 늘었으나, 결함 검출 효율은 오히려 감소하는 경향을 보였다. 특히 GPT-5는 GPT-3보다 더 많은 오류를 탐지했으나, GPT-7은 효율성이 낮아졌다.

반면, GPT+전략 지시 그룹은 연구자 설계와 동일하게 7개의 테스트 케이스를 사용하였음에도 불구하고, 연구자 설계보다 더 많은 오류를 검출하여 가장 높은 오류 탐지 성능을 보였다. 이는 등가 분할, 경곗값 분석, 비정상 입력, 조건 분기 커버리지, 오라클 기반 평가 등을 명시적으로 지시한 프롬프트가 AI의 테스트 생성 품질을 크게 향상시켰음을 의미한다. 연구자 설계 그룹은 GPT+전략 지시보다는 낮은 성능을 기록했지만, 여전히 단순 GPT 생성 그룹보다 높은 오류 검출력을 보였다.

표 1은 각 테스트 케이스 그룹이 35개의 삼각형 판별 프로그램에 대해 Pytest 자동 실행을 수행한 결과를 요약한 것이다. 총 테스트 실행 수는 '테스트 케이스 수 × 35개 함수'로 계산되며, 오류 검출은 예상 결과와 실제 결과가 일치하지 않은 경우로 정의하였다.

학생들이 직접 작성한 테스트 케이스는 총 75회 실행에서 13건의 오류만을 검출하여, GPT 기반 케이스보다 낮은 결함 검출 효율을 보였다. 이는 학생 작성 테스트가 정상 입력이나 단순한 조건에 치중되는 경향이 있어 다양한 오류 상황을 충분히 탐지하지 못했음을 보여준다. 구체적으로, 학생 작성 테스트 케이스는 정상 입력이 전체 61개(71%)로 편중되었으며, 무효 입력은 14개에 불과하였다. 일부 학생은 테스트 케이스를 생성하지 못해, 이 경우 프로그램 실행 과정에서 자동으로 건너뛰도록 처리하였다.

오류 검출 성능은 학생 작성, GPT 단순 지시, 연구자 설계, GPT+전략 지시 순으로 나타났다. 이는 프롬프트 엔지니어링을 통해 전략을 명시적으로 지시할 경우 AI가 연구자 수준을 넘어서는 테스트 품질을 생성할 수 있음을 보여준다.

4. 테스트 결함 검출 효율 기반 정량 분석

본 절에서는 테스트 결함 검출 효율을 기반으로 정량 분석 결과를 논의한다. 테스트 결함 검출 효율 비교는 표 2에 제시하였다.

GPT-3, GPT-5, GPT-7, GPT+전략 지시, 본 연구 설계가 검출한 오류 수는 각각 35, 60, 67, 91, 85였으며, 이에 따른 테스트 결함 검출 효율은 11.67, 12.00, 9.57, 13.00, 12.14로 집계되었다.

동일한 7개의 테스트 케이스를 사용했음에도 GPT+전략 지시 그룹이 13.00으로 가장 높은 효율을 보였다. 이는 연구자 설계(12.14)보다 우수한 성능으로, 프롬프트에 전략적 지시를 추가하는 것이 테스트 품질 향상에 기여함을 보여준다.

학생들이 직접 작성한 테스트 케이스는 결함 검출 효율이 0.17로, GPT 기반 테스트 케이스에 비해 현저히 낮았다. 이는 정상 입력 위주의 설계와 일부 미작성 사례로 인해 다양한 오류 상황을 포착하지 못했기 때문이다.

입력 구성 측면에서 연구자 설계와 GPT+전략 지시 케이스는 정상 입력, 비정상 입력, 경곗값, 삼각형 성립 불가 조건을 모두 포함하였다. GPT+전략 지시는 연구자 설계와 동일한 전략을 따르면서, 일부 입력이 다양하게 분포하여 효율 향상으로 이어졌다. 반면 GPT 단순 지시 케이스는 특정 유형에 편중되거나 유사 입력이 반복되는 문제가 있어 효율이 제한되었다. 학생 작성 케이스는 정상 입력에 크게 치중되어 상대적으로 오류 탐지 범위가 좁았다.

표 2. 테스트 케이스 그룹별 결함 검출 효율 비교

테스트 케이스 수	총 오류 수	테스트 결함 검출 효율 (Faults per Test)
3개 (GPT-3)	35	11.67
5개 (GPT-5)	60	12.00
7개 (GPT-7)	67	9.57
7개 (GPT+전략 지시)	91	13.00
7개 (본 연구 설계)	85	12.14
학생 작성 케이스	13	0.17

Ⅳ. 논 의

1. 테스트 구성 전략의 중요성

AI 생성 테스트 케이스 비교 결과는 단순히 테스트 케이스의 수를 늘리는 방식으로는 오류 검출 성능을 안정적으로 높일 수 없음을 보여준다. 가장 중요한 요인은 입력의 질적 구성과 전략적 설계였다. GPT+전략 지시 그룹은 연구자 설계와 동일한 7개의 테스트 케이스를 사용하였음에도 불구하고, 연구자 설계보다 높은 오류 검출과 효율을 기록하였다. 이는 프롬프트에 전략적 지시를 포함하는 것이 AI 테스트 품질을 결정짓는 핵심 변수임을 실증적으로 입증한 결과이다.

학생 작성 케이스는 정상 입력에 편중되어 예외 상황을 충분히 반영하지 못했으며, 일부는 테스트를 작성하지 않아 실행 과정에서 제외되었다. 이러한 불균형과 제한성은 입문자 테스트의 특성을 보여준다. 반면 전략적으로 설계된 테스트 케이스는 적은 수라도 입력의 다양성을 갖추고 있어 높은 결함 검출 효율을 보였다.

오류 유형 분포는 교육적 시사점도 제공한다. 예를 들어, 정삼각형 오류는 조건 분기 설계의 미비나 대칭 구조 이해 부족을 시사하고, 무효 입력 오류는 비정상 입력 및 경곗값 처리의 미흡함을 드러낸다. 따라서 오류 유형별 분석은 단순 수치 이상의 의미를 가지며, 학습자의 사고 편향과 전략적 약점을 진단하는 정성적 분석 도구로 활용될 수 있다.

자동 생성 GPT 테스트 케이스 그룹의 경우, 테스트 케이스 수와 효율 간에 뚜렷한 상관관계가 나타나지 않았다. GPT-5는 GPT-3보다 약간 높은 효율을 보였는데, 이는 일부 중복 입력이 줄어든 영향으로 해석할 수 있다. 반면 GPT-7은 입력 중복과 편중으로 인해 효율이 낮아졌다. 특히 동일한 7개 조건을 사용한 본 연구 설계와 GPT-7을 비교했을 때 큰 격차가 확인되었으며, 이는 테스트 효율이 단순한 개수보다 전략적 목적과 일관성에 더 크게 좌우됨을 보여준다.

2. TDD 활용의 교육적 시사점

본 연구에서 사용한 테스트 결함 검출 효율은 교육 현장에서 학습자의 테스트 설계 역량을 진단하는 정량 지표로 활용할 수 있다. 동일 과제, 동일 실행 조건에서 AI 기반 테스트와 학습자의 테스트 케이스의 테스트 결함 검출 효율을 비교하면, 전략적 설계 수준과 입력 커버리지의 차이

를 파악하고 즉각적인 피드백 자료로 제시할 수 있다. 수업 적용 시 단계별 테스트 결함 검출 효율(실습 전, 중, 후)을 추적하여 학습 진전을 계량화할 수 있고, 유형별 테스트 결함 검출 효율로 세분 평가도 가능하다.

GPT+전략 지시 그룹의 결과는 단순한 AI 활용만으로는 충분하지 않으며, 학습자가 전략을 어떻게 지시·수정·분석하느냐에 따라 AI가 연구자 수준을 넘어서는 성능을 발휘할 수 있음을 보여준다. 따라서 교육 현장에서는 프롬프트 엔지니어링을 도입하여 학습자가 전략적 설계를 직접 실습할 수 있도록 설계하는 것이 효과적이다.

이러한 교육적 효과는 다수의 선행 연구에서도 확인된 바 있으며[2][3], 본 연구는 이를 정량 지표와 AI 활용 결과를 통해 확장·실증하였다. 특히 TDD의 네 단계(테스트 작성-실행-실패 확인-수정) 반복 과정은 논리 구조화, 즉각적 피드백, 오류 인식, 반복 학습을 유도하는 사고 중심 학습 전략으로 작용한다.

3. TDD 교육에 AI 도구 활용

2절에서 설명한 것과 같이, TDD 교육에는 AI를 적용할 수 있다. 이에 우리는 입문자에게 AI 도구 활용이 적합한지 확인하기 위해, ChatGPT를 활용한 테스트 케이스 자동 생성의 장단점을 분석하였다. 분석 결과를 바탕으로, ChatGPT 기반 테스트 자동 생성의 교육적 활용 가능성, 자동 생성 테스트의 한계와 교육적 대응 전략, 그리고 TDD 기반 교육에서 AI 도구의 활용 전략 주제에 대해 논의한다.

(1) ChatGPT 기반 테스트 자동 생성의 교육적 활용 가능성

ChatGPT가 자동으로 생성한 테스트 케이스는 입문자가 간과하기 쉬운 경곗값이나 삼각형 성립 조건 위반과 같은 비정상 입력을 일부 포함하고 있어, 코드 결함을 조기에 유도하는 데 효과적이었다. 이는 AI 기반 테스트가 테스트 다각화를 위한 출발점으로 활용될 수 있음을 시사하며, 특히 TDD 교육에서는 이를 정답으로 수용하기보다 학습자의 비판적 사고를 유도하는 자료로 활용하는 전략이 바람직하다.

실제로 GPT가 생성한 3-5개의 테스트 케이스는 일부 수작업 테스트보다 높은 테스트 결함 검출 효율을 기록하였으며, 입문자 실습에서 전략적 테스트 설계 방향성을 설정하는 데 기여할 수 있다. 이러한 결과는 AI가 제공하는 자동 생성 테스트가 교육 초기 단계에서의 사고 확장을 촉진하는 도구로 기능할 수 있음을 보여준다.

(2) 자동 생성 테스트의 한계와 교육적 대응 전략

AI 도구의 활용은 교육적 가능성을 지니고 있으나, 그에 따른 몇 가지 한계도 동시에 고려되어야 한다. AI가 항상 최적의 테스트 케이스를 생성하는 것은 아니며, 실제 생성된 결과에서는 조건 편중, 입력 중복, 경곗값 누락 등 구조적 결함이 반복적으로 관찰되었다.

표 3은 ChatGPT가 자동 생성한 테스트 케이스의 입력 구성 특성을 비교한 것이다. 각 그룹의 테스트가 입력 커버리지 측면에서 어떤 차이를 보였는지를 정량적으로 보여준다. GPT-3는 정상 입력 위주로 편중되어 있었으며, 비정상 입력이나 경곗값 입력이 거의 포함되지 않았으며, GPT-5는 일부 비정상 입력을 포함했으나 여전히 경곗값 입력은 부족하였고, 중복 입력도 다수

표 3. ChatGPT 자동 생성 테스트 케이스 입력 구성 비교

구분	테스트 수	정상 입력 포함 여부	비정상 입력 포함 여부	경곗값 포함 여부	입력 중복 정도	입력 다양성
GPT-3	3개	○	△	×	높음	낮음
GPT-5	5개	○	○	△	중간	중간
GPT-7	7개	○	○	△	낮음	높음

※ 기호 설명: ○ = 포함됨 / △ = 일부 포함 (예: 미포함 또는 1개) / × = 포함되지 않음

※ 입력 중복은 낮을수록, 다양성은 높을수록 바람직함.

존재했다. GPT-7은 상대적으로 가장 다양한 입력을 포함했지만, 극단값이나 예외 입력 등 고차원적 조건 반영은 여전히 미흡하였다. GPT-3는 입력 중복이 높고 입력 다양성은 낮은 경향을 보인 반면, GPT-7은 중복이 낮고 커버리지가 넓은 테스트 구성을 보여주었다. GPT-7은 무효 입력 오류에서 20.9%의 탐지율을 기록한 반면, GPT-3는 8.6%에 그쳤다.

AI 기반 테스트 케이스는 교육적으로 유용한 출발점이 될 수 있으나, 그 자체를 최종 해답으로 수용하기보다는 학습자의 비판적 분석 대상으로 활용되어야 한다. 특히 TDD 기반 교육에서는 "이 테스트는 충분한가?", "빠진 입력은 없는가?"와 같은 질문을 통해 학습자가 AI 결과를 능동적으로 점검하고 보완하는 활동이 병행되어야 한다. 이러한 실습은 학습자의 전략적 사고력, 입력 및 테스트 커버리지 설계 능력을 강화한다.

Ⅲ. 결 론

본 연구는 Python 입문자를 대상으로 한 TDD 기반 실습 환경에서 테스트 케이스 구성 전략이 오류 검출 성능과 교육 효과에 미치는 영향을 분석하며, AI 활용 전략의 필요성을 강조한다. 단순히 테스트 수를 늘리는 것보다 입력 조건의 다양성과 전략적 설계가 더 큰 영향을 미쳤고, GPT에 전략적 설계 조건을 명시했을 때 연구자 설계보다도 우수한 오류 검출 효율을 보였다.

AI 도구는 학습 초기에는 참고 예시로 제공하되, 최종 설계는 학습자가 직접 수행하도록 유도하고, AI 결과를 분석·보완하는 활동을 통해 전략적 설계 역량을 내면화하는 것이 바람직하다. AI를 TDD 사이클에 통합하여 반복적 보완 활동을 촉진하면, 학습자는 자연스럽게 오류 유도력과 커버리지를 고려하는 역량을 기르게 된다.

향후 복잡한 문제를 대상으로 GPT 단순 생성, 연구자 설계, GPT+전략 지시 방식의 효과를 비교해 소프트웨어 공학적 관점에서 AI 기반 테스트 설계의 확장 가능성을 검증할 예정이다.

REFERENCES

[1] Beck, K., Test-Driven Development: By Example, Addison-Wesley, pp.1-220, 2003.

[2] H. Erdogmus, M. Morisio, and M. Torchiano, "On the effectiveness of the test-first approach to programming," *IEEE Trans. Softw. Eng.,* vol. 31, no. 3, pp. 226-237, Mar. 2005.

[3] S. Edwards, "Using test-driven development in the classroom: Providing students with automatic, concrete feedback on performance," *Proc. Int. Conf. Educ. Inf. Syst., Technol. Appl. (EISTA),* pp. 421-426, Orlando, FL, USA, 2003.

[4] C. Kaner, "Exploratory testing," *Softw. Test. Qual. Eng.,* vol. 3, no. 1, pp. 84-87, 2002.

[5] M. Basili, L. Briand, and W. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Trans. Softw. Eng.,* vol. 22, no. 10, pp. 751-761, Oct. 1996.

[6] V. Hellendoorn, C. Treude, C. Le Goues, and D. E. Perry, "Global software engineering in the age of AI," *Proc. IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE),* pp. 856-867, Luxembourg, 2021.

[7] X. Gao, Q. Luo, M. Zhang, and Z. Huang, "Can ChatGPT write effective test cases? An empirical study," arXiv preprint arXiv:2305.13792, 2023.

저 자 소 개

손새봄(준회원)

2026년 한남대학교 컴퓨터공학과 학사 졸업 예정.

<주관심분야 : 소프트웨어 테스팅, 소프트웨어 품질보증, 소프트웨어 테스트 자동화>

송지영(종신회원)

2014년 이화여자대학교 컴퓨터공학과 학사 졸업.
2016년 한국과학기술원 전산학부 석사 졸업.
2022년 한국과학기술원 전산학부 학과 박사 졸업.

<주관심분야 : 소프트웨어공학, 소프트웨어 테스팅, 소프트웨어 모델 검증, 프로젝트 관리, 시스템 오브 시스템즈, IoT, CPS>