

<http://dx.doi.org/10.17703/JCCT.2026.12.2.155>

JCCT 2026-3-15

생성형 AI를 이용한 소프트웨어 특허 문서로부터 가장 작은 기능적 요소 추출 메커니즘

A Mechanism for Extracting the Smallest Functional Elements from Software Patent Documents using Generative AI

장우성, 김영철, 박현석

Woosung Jang*, R. Young Chul Kim**, and Hyunseok Park***

요약 소프트웨어를 검증하기 위해서는 다양한 측면에서 많은 테스트를 수행하는 것이 중요하다. 역공학 기반 테스트는 중간 산출물들로부터 테스트 케이스를 생성 및 실행한다. 하지만 다양한 중간 산출물 중 소프트웨어 특허로부터 테스트 케이스 생성 연구는 미비하다. 특허 문서는 일반적인 요구사항 명세서와는 다른 형태를 가지기 때문에 기존 요구사항 분석 기법과는 다른 분석 방법이 필요하다. 또한 특허 문장의 해석, 구성요소 분류 등은 자동화가 어렵다. 기존 문제의 해결을 위해, 우리는 생성형 AI를 사용하여 소프트웨어 특허 문장으로부터 가장 작은 기능적 요소를 추출하는 메커니즘을 제안한다. 1) 소프트웨어 특허 문서로부터 텍스트 데이터를 추출한다. 2) 추출된 데이터를 분류하여 알고리즘이 포함된 파트를 식별한다. 3) 문장을 정규화한다. 4) 소프트웨어 기능을 설명하는 문장을 식별한다. 5) 문장을 단문화 한다. 6) 단문에서 가장 작은 기능적 요소를 식별한다. 이 방법은 특허 자연어 문장으로부터 기능적 요소를 분류 및 식별하여 테스트 케이스 생성을 위한 기반을 마련할 수 있다. 테스트 케이스가 중간 산출물로부터 자동 생성되기 때문에 테스트 비용이 감소될 수 있다.

주요어: 소프트웨어공학, 특허, 생성형 AI, 자연어 분석, 원인-결과 그래프, 요구사항명세

Abstract Software verification requires extensive testing across various aspects. Reverse engineering-based testing generates and executes test cases from intermediate outputs. These artifacts include software patents. However, research on test case generation from software patents is very few. Because patent documents have a different format than typical requirements specifications, they require a different approach than traditional analysis techniques. Also processes such as patent sentence interpretation and component classification are difficult to automate. To solve this problem, we proposes a mechanism that extracts the smallest functional elements from software patent sentences using generative AI. 1) Text data is extracted from software patent documents. 2) The extracted text is classified, and text containing algorithms is identified. 3) The sentences are normalized. 4) Sentences describing software functions are identified. 5) The sentences are simplified. 6) The simplified sentences are converted into the smallest functional components. This method classifies and identifies functional elements from patent natural language sentences. Furthermore, it can generate basic model for test case generation. Testing costs can be reduced because test cases are automatically generated from intermediate outputs.

Keywords: Software engineering, patents, generative AI, natural language analysis, cause-and-effect graphs, requirements specifications

*정회원, 한양대학교 정보시스템학과 박사후연구원 (제1저자)

**정회원 홍익대학교 소프트웨어융합학과 교수 (참여저자)

***정회원, 한양대학교 정보시스템학과 교수 (교신저자)

접수일: 2025년 12월 26일, 수정완료일: 2026년 1월 19일

게재확정일: 2026년 1월 31일

Received: December 26, 2025 / Revised: January 19, 2026

Accepted: January 31, 2026

***Corresponding Author: hp@hanyang.ac.kr

Dept. of Information System, Hanyang Univ Korea

I. 서 론

소프트웨어(SW) 엔지니어링에서 SW 검증은 중요한 이슈이다. 역공학 기반 테스트는 SW 개발 산출물로부터 설계도 또는 테스트 케이스를 생성하여 SW를 검증할 수 있다. 역공학 기반의 설계 모델을 실제 설계 모델과 비교할 수 있고, 역공학 기반의 테스트 케이스를 통해 요구사항 명세를 검증할 수 있다[1]. 많은 테스트를 수행할수록 SW가 견고해지기 때문에, 다양한 측면에서 테스트를 수행하여 SW를 검증해야 한다[2,3]. SW 산출물 중 SW 특허는 일반적으로 SW의 핵심 기능을 기술한다. 하지만 특허 문서로부터 테스트 케이스 생성은 어렵다. 특허에서 사용되는 문장은 일반적인 문장과 구성이 달라 자동 해석이 어렵고, 특허 내 기능적 요소만 발췌하여 테스트 케이스를 생성하기가 어렵다.

문제의 해결을 위해, 본 논문은 생성형 AI를 이용한 SW 특허 문서로부터 가장 작은 기능적 요소 추출 메커니즘을 제안한다. 특허 PDF 문서로부터 텍스트 데이터를 인식하여 세부 항목을 분류하고, 알고리즘이 기술된 항목만을 추출한다. 문장 정규화 과정을 통해 특허 문장을 일반 문장으로 변환한다. 변환된 문장 중 복합절로 구성된 문장을 단문으로 변환한다. 단문화 된 문장들로부터 가장 작은 기능적 요소를 추출한다. 추출된 요소들은 Cause-Effect Graph의 노드로 정의된다. Cause-Effect Graph는 테스트 케이스를 생성할 수 있는 모델이다. Cause-Effect Graph 기반 테스트 케이스는 최소한의 테스트 케이스 수로 100% 테스트 커버리지를 달성할 수 있다[2].

본 논문의 구조는 다음과 같다. 2장은 관련 연구를 언급한다. 3장은 생성형 AI를 이용한 SW 특허 문서로부터 가장 작은 기능적 요소 추출 메커니즘을 언급한다. 4장은 실험 및 결과를 언급한다. 마지막으로 결론을 언급한다.

II. 관련 연구

1. Cause-Effect Graph

Cause-Effect Graph는 SW 요구사항 문장을 논리적 구조로 모델링하고, 테스트 케이스 생성의 기반을 마련한다[4]. 자연어 문장에서 가장 작은 기능적 요소들을 식별하고, 식별된 기능들을 원인과 결과 관계로 연결한다. Myers[2]에 의하면, 가장 작은 기능적 요소는 요구사항 명세에서 가장 작은 업무 단위를 의미한다.

그림 1은 Cause-Effect Graph의 노드와 연결 관계를 보여준다. 노드는 요구사항 문장에서 추출된 가장 작은 기능적 요소이다. 노드 간 연결 관계에서 좌측 노드는 원인이 되는 기능이고, 우측 노드는 결과가 되는 기능이다. 원인과 결과 관계는 Identity, NOT, OR, AND 관계를 포함한다. Identity는 원인이 true이면 결과가 true이다. NOT은 원인이 false면 결과가 true이다. OR는 여러 개의 원인 중 하나라도 true라면 결과가 true이다. AND는 여러 개의 원인이 모두 true이면 결과가 true이다.

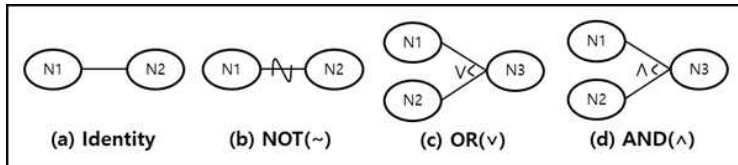


그림 1. Cause-Effect Graph의 노드와 연결관계

Figure 1. Node and relationships of Cause-Effect Graph

그림 2는 요구사항 문장으로부터 Cause-Effect Graph 생성 예제를 보여준다. 문장에서 3개의 작은 기능적 요소를 식별할 수 있다. 각 요소는 Cause-Effect Graph의 노드로 표현된다. A 노드와 B 노드는 원인 노드이다. C 노드는 결과 노드이다. A 노드와 B 노드 중 하나라도 true이면 C 노드가 true이다.

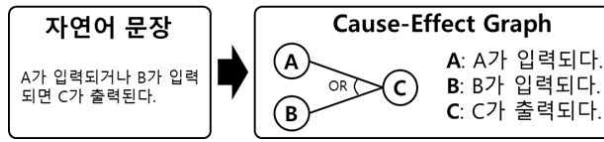


그림 2. Cause-Effect Graph 예제
Figure 2. Cause-Effect Graph Example

2. 한국어 말뭉치 정규화

한국어 말뭉치 정규화[5]는 문장의 구조를 좀 더 명료하게 변환할 수 있다. 한국어에서 문장 구성 성분 간의 관계를 보면 주로 조사에 의해 주어, 동사 등의 성분이 정의된다. 말뭉치 정규화는 격이동 복원을 통한 조사들의 변경과 생략된 주어 복구를 통해 문장의 구조를 명료하게 변환한다. 자연어 문장을 정규화하여 생성형 AI로 분석하면 좀 더 높은 정확도를 얻을 수 있다[6].

표 1은 말뭉치 정규화 과정의 예제를 보여준다. 말뭉치 정규화는 피동문을 능동문으로 변환하고, 사동문을 주동문으로 변환하고, 절 분리 시 주어가 생략된 절의 주어를 복구한다. 피동문은 주체가 동작 또는 상태의 변화를 입는 것을 의미한다. 이것은 타 행위자에 의해 주체의 동작이나 상태가 변환된다. 능동문은 스스로의 힘으로 행하는 행위나 동작을 의미한다. 사동문은 남에게 어떤 동작을 부여하는 것을 의미한다. 주동문은 동작이나 행위를 자기 스스로 행하는 것을 의미한다.

표 1. 말뭉치 정규화 예제
Table 1. Corpus normalization example

분류	변환 전	변환 후
피동문을 능동문으로 변환	모듈에 의해 문장이 입력된다.	모듈이 문장을 입력하다.
사동문을 주동문으로 변환	서버가 클라이언트에게 정보를 처리하게 하다.	클라이언트가 서버에 의해 정보를 처리하다.
절 분리 시 생략된 주어 복구	사용자는 계산 정보를 저장하려고 로그인한다.	사용자가 계산 정보를 저장한다. 사용자가 로그인한다.

2. 프롬프트 엔지니어링

프롬프트 엔지니어링은 대규모언어모델(LLM)을 효율적으로 사용할 수 있도록 도와주는 프롬프트 기술이다. 본 논문은 Chain of Thought(CoT) 프롬프트[7]와 후카즈식 프롬프트[8]를 사용하여 특히 문장에서 Cause-Effect Graph의 노드를 추출한다. CoT 프롬프트는 중간 추론 단계를 통해 복잡한 추론을 수행할 수 있도록 지원한다. 복잡한 문제를 해결할 때 각 단계를 명확히 제시하고, 논리적인 순서로 문제를 해결한다. 예시를 제시하여 추론의 정확도를 증가시킬 수 있다. 후카즈식 프롬프트는 생성형 AI와 상호작용을 최적화하는 전략을 제시한다. 지침, 제약조건, 입력문 등을 명시하여 생성형 AI가 출력할 결과에 대한 다양한 조건들을 명시한다. 표 2는 각 프롬프트 기법의 예제를 보여준다.

표 2. 프롬프트 엔지니어링 예제
Table 2. Prompt engineering example

CoT 프롬프트	후카즈식 프롬프트
문제: 사과가 10개 있었다. 3개를 먹었고 2개를 더 샀다. 사과는 총 몇 개가 되었나? 답변: 10개에서 3개를 먹었으니 (10-3=7)개가 남았다. 2개를 더 샀으니 (7+2=9)개가 되었다. 정답은 9개이다. 문제: 식당에는 사과가 23개 있었다. 점심 식사에 20개를 사용하고, 6개를 더 구매했다면, 현재 몇 개의 사과를 가지고 있는가? 답변: [생성형 AI의 답변을 출력]	#지침: 당신은 소프트웨어 관련 강의 교안을 만든다. 다음의 제약조건 및 입력문을 기반으로 대답하시오. #제약조건: - 소프트웨어 관련 용어 외에는 일반적인 용어를 사용해야 한다. - 문장을 짧고 명확하게 구성한다. [이하 생략] #입력문: 디자인페턴 수업에 사용할 교안을 만든다. 디자인페턴의 개요를 설명한다. [이하 생략] #출력문:

III. 생성형 AI를 이용한 SW 특허 문서로부터 가장 작은 기능적 요소 추출 방법

그림 3은 전체 프로세스를 보여준다. 주황색 단계들은 본 논문의 연구를 의미하고, 회색 단계들은 향후 연구를 의미한다. SW 특허 입력 단계는 PDF 파일을 입력한다. 광학 문자 인식 단계는 PDF로부터 텍스트를 식별하고 저장한다. 문장 보정 단계는 광학 문자 인식 단계에서 잘못 인식되는 문자들을 보정한다. 문장 분류 단계는 특허 문서의 내용을 분류한다. 특허 내용 중 과제의 해결 수단, 발명을 실시하기 위한 구체적인 내용을 추출한다. 문장 정규화 단계는 특허 문장 구조를 일반적인 문장 구조로 변환한다. 기능을 설명하는 문장 식별 단계는 정규화 된 문장에서 SW 기능들의 실행 순서를 식별한다. 만약 정규화된 내용이 SW 기능 실행 순서를 포함하지 않고, 모듈 구성요소를 설명만 하는 문장이라면, 해당 문장은 배제된다. 문장의 단문화 단계는 2개 이상의 조건절이나 접속절이 포함된 문장을 단문으로 나눈다. 노드 정의 단계는 단문화된 문장들을 Cause-Effect Graph의 노드로 정의한다. 조건절과 결과절은 Identity 관계의 노드들로 표현된다. 결과절이 부정적이라면 NOT 관계로 표현된다. 접속절의 선행절과 후행절은 AND 또는 OR 연결관계로 표현된다.

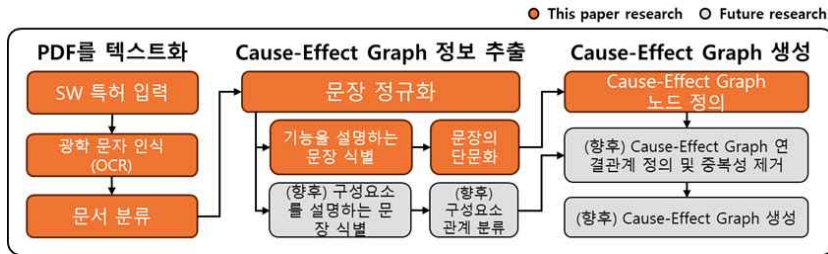


그림 3. 전체 프로세스

Figure 3. Our Process

프로세스에서 가장 핵심 단계인 문장 정규화 단계에서 수행되는 내용은 표 3과 같다. 각 구성요소들은 프롬프트 엔지니어링 기법을 통해 프롬프트로 표현된다. 특허에서만 쓰이는 단어를 불용어 처리한다. 문장의 표현을 단순화한다. 문장을 조건문 위주로 표현한다. 문장 구조를 정형화한다. 말뭉치 정규화를 수행한다. 생성형 AI가 잘못된 문장을 생성하면 향후 잘못된 테스트 케이스가 생성될 수 있기 때문에[9] 할루시네이션 방지를 위한 조치를 취한다. 정규화된 문장은 수동 검증 과정을 거친다.

표 3. 문장 정규화 단계의 구성요소

Table 3. Components of sentence normalization

구성요소	설명	예시
불용어 리스트 작성	특허 문서에서만 주로 사용되는 단어를 불용어 처리한다.	“일 실시예”, “상기” 등의 단어를 불용어 처리
문장 단순화	유사한 의미를 가지는 행위를 단순화한다.	“할 수 있다”, “해야 한다”를 “하다”로 변경
조건문으로 표현	의미적으로 연결되는 기능은 조건문으로 변경한다.	“입력창 로딩이 완료된다. 다음으로 아이디를 입력한다.”를 “입력창 로딩이 완료되면, 아이디를 입력한다.”로 변경
문장 구조의 정형화	모든 문장은 반드시 주어와 동사를 포함하고, 최대한 목적어를 포함한다.	“모듈이 로딩되면, 텍스트를 전송한다.”에서 후행절을 “텍스트가 모듈에게 전송된다.”로 변경
말뭉치 정규화	모든 문장을 능동문, 주동문으로 표현한다.	“서버가 클라이언트에게 정보를 처리하게 하다.”를 “클라이언트가 서버에 의해 정보를 처리하다.”로 변경
할루시네이션 방지	원문에 없는 기능이나 문장을 추가 및 변경하지 않는다.	원문이 “로그인 화면에 아이디를 입력한다.”라면, “로그인 화면에 아이디와 비밀번호를 입력한다.”와 같이 원문에 없는 내용이 추가되는 것을 방지

표 4는 문장 정규화 단계를 CoT 프롬프트 문장으로 표현한 결과의 일부이다. 표 6은 문장 정규화 단계를 후카츄식 프롬프트로 표현한 결과의 일부이다. 표 6은 생성형 AI를 통해 특허 문장을 일반적인 문장으로 변환 후 가장 작은 기능적 요소를 추출하여 노드로 정의한 예제이다.

표 4. CoT 프롬프트 문장의 일부

Table 4. Part of CoT Prompt Sentence

아래 정보들을 바탕으로 특허 문서 내 문장들을 일반적인 문장 형태로 변환한다.
입력되는 문장들은 소프트웨어 모듈의 정의, 기능, 연관관계를 설명한다. 일반적인 문서에서 사용되지 않고 특허에서만 주로 사용되는 단어를 제거한다. 이때 단어 자체가 가지는 의미가 없어야 한다. 예를 들어 "상기" 단어는 제거된다. 세미콜론은 제거된다. 일 실시예는 제거된다. 변환되는 문장은 주어, 동사를 반드시 포함해야 하고, 목적어를 최대한 포함해야 한다. 목적어는...[중간 생략]
문제: "상기 서비스 소비 계획을 수립하는 단계는, 서비스 기반의 차량-사물 통신 환경..." [이하 생략]

표 5. 후카츄식 프롬프트 문장의 일부

Table 5. Part of Fukatsu Prompt Sentence

#지침:
당신은 특허 문서 내 문장을 일반적인 문장 형태로 변환해야 한다.
아래의 제약조건을 기반으로 특허 문장이 입력된 입력문을 일반적으로 사용되는 문장으로 변환하여 출력하시오.
#제약조건:
- 입력되는 문장들은 소프트웨어 모듈의 정의, 기능, 연관관계를 설명한다.
- 변환되는 문장은 반드시 주체가 포함되어야 한다. [이하 생략]

표 6. 특허 원본 문장으로부터 가장 작은 기능적 요소 추출 예제

Table 6. Example of extracting the smallest functional element from the original patent sentence

종류	문장
특허 원본 문장	상기 스마트 게이트웨이가 동작되면 IoT 엔진에서 스마트 게이트웨이의 Public IP, Local IP, Mac Address 등의 정보를 식별하고, 사용자 아이디와 상기 식별된 정보를 인증 서버에 전송하면, 인증 서버는 전송받은 데이터를 데이터베이스에 저장할 수 있다.
생성형 AI에 의해 변환된 문장	스마트 게이트웨이가 동작되면 IoT 엔진이 스마트 게이트웨이의 Public IP, Local IP, Mac Address 등의 정보를 식별한다. IoT 엔진이 사용자 아이디와 식별된 정보를 인증 서버에 전송하면 인증 서버는 전송받은 데이터를 데이터베이스에 저장한다.
가장 작은 기능적 요소 추출 (단문화)	A: 스마트 게이트웨이가 동작된다. B: IoT 엔진이 스마트 게이트웨이의 Public IP, Local IP, Mac Address 등의 정보를 식별한다. C: 사용자 아이디와 Public IP, Local IP, Mac Address 등의 정보를 인증 서버에 전송한다. D: 인증 서버가 전송 받은 데이터를 데이터베이스에 저장한다.

IV. 연구 결과

그림 4는 실험 환경의 설계를 보여준다. PDF Parser는 PyPDF2 라이브러리를 사용하여 PDF를 텍스트로 변환한다. Sentence Classifier는 특허 내용을 분류한다. Sentence Normalizer는 생성형 AI를 사용하여 특허 문장에서 가장 작은 기능적 요소를 추출한다. Function Description Identifier는 생성형 AI를 사용하여 문장이 SW 기능 실행 순서 여부를 식별한다. Cause-Effect Graph Node Generator는 가장 작은 기능적 요소를 노드로 변환한다.

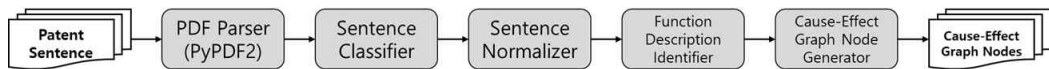


그림 4. 실험 환경 설계

Figure 4. Experimental environment design

실험 환경은 텍스트 번역, 생성, 요약 등의 다양한 자연어 처리에 특화된 4개 생성형 AI 서비스를 이용하여 특허 문장을 일반 문장으로 변환하고, 변환 정확도를 비교한다[10]. 전문가가 작성한 예상값과 실제 출력 값을 비교하여 정답율을 계산한다. 실험 데이터 표본 선정 방법은 다음과 같다. 특허 CPC 리스트에서 소프트웨어와 관련된 항목들을 분류하여 50개를 선별한다. 그리고 각 CPC와 관련된 특허들을 수집하고, 소프트웨어 알고리즘이 포함된 특허

문서를 수동 선별한다. 선별된 문서는 PDF Parser를 통해 한글이 인식된다. 잘못 인식된 한글은 없었지만, 문단과 문단 사이에 항목 번호가 삽입되거나, 잘못된 줄바꿈 등의 문제가 발생되었다. 이것은 예외처리를 통해 수정된다.

표 7은 소프트웨어 특허 문서의 청구항 30개를 후카츠식 프롬프트와 CoT 프롬프트가 적용된 ChatGPT[11], Claude[12], Gemini[13], Copilot[14]로 분석한 결과이다. ChatGPT는 GPT-5.2, Claude는 Claude Sonnet 4.5, Gemini는 Gemini 2.5 Flash, Copilot은 GPT-4 모델을 사용한다. 후카츠식 프롬프트의 경우 ChatGPT와 Claude는 모든 청구항에서 기능적 요소의 일부가 누락되었다. Gemini는 80% 문장이 모듈 구성요소만을 설명하고, 모듈 간 기능 흐름 설명을 누락하였다. Copilot은 20% 문장에서 주어가 식별되지 않았다. CoT 프롬프트의 경우 ChatGPT, Gemini, Copilot이 문장에서 작은 기능적 요소를 완벽하게 추출하였고, Claude는 일부 문장이 누락되었다.

표 7. 실험 결과

Table 7. Experimental results

	ChatGPT (GPT-5.2)	Claude (Claude Sonnet 4.5)	Gemini (Gemini 2.5 Flash)	Copilot (GPT-4)
후카츠식 프롬프트	0%	0%	20%	60%
CoT 프롬프트	100%	90%	100%	100%

V. 결론

특히 문장에서 가장 작은 기능적 요소를 추출하기 위해 생성형 AI를 사용하여 문장을 정규화하고, 기능의 흐름이 포함된 문장을 식별하였다. 문장을 정규화하면 생성형 AI의 문장 분석 정확도를 증가시킬 수 있고, 프롬프트 엔지니어링을 적용하면 생성형 AI의 학습 정확도를 증가시킬 수 있다. 이 방법을 통해 좀 더 높은 정확도로 특허 문장을 분석하여 기능을 추출할 수 있다. 또한 본 방법을 구조적 요구사항 문장에 적용하여 요구사항 문장에서 기능적 요소 추출을 수행할 수 있다. 하지만 비기능적 요구사항에 적용할 경우 높은 할루시네이션을 유발할 수 있다.

실험 결과, 전반적으로 CoT 프롬프트가 높은 정확도를 보여준다. 특히 ChatGPT의 경우 전문가가 수동적으로 생성한 문장과 비교하여 가장 자연스러운 문장을 생성하였다. 하지만 좀 더 많은 실험 데이터를 통한 정확도 측정 및 검증이 필요하다. 또한 본 논문에서 제안하는 방법은 기능이 추출된 모든 문장으로부터 Cause-Effect Graph 노드를 생성하지 못한다. 노드를 생성하기 위해서는 문장에 원인결과 결과절이 포함되어야 하기 때문이다. 이외에도 프롬프트 문장만을 이용한 학습을 수행하기 때문에 모델의 파라미터 튜닝이나 추가적인 모델 학습이 불가능한 한계를 가진다.

향후 연구로써, 원인결과 결과절이 포함되지 않은 문장으로부터 Cause-Effect Graph 노드를 생성하고, 노드 간 연결 관계를 문장으로부터 추출하는 연구를 수행할 예정이다.

References

- [1] S. Y. Moon, and R. Y. C. Kim, "Cost Estimation and Validation based on Natural Language Requirement Specifications," International Journal of Internet, Broadcasting and Communication (IJIBC), Vol. 15, No. 2, pp. 218-226, 2023.
- [2] G. J. Myers, "The Art of Software Testing", Jogn Wiley & Sons, Inc., 2004.
- [3] W. Shin, "A Study on Test-Driven Development Method with the Aid of Generative AI in Software Engineering," International Journal of Internet, Broadcasting and Communication (IJIBC), Vol. 16, No.4, pp.194-202, 2024.
- [4] G. E. Mogyorodi, B. Math., M.B.A., "Requirements-Based Testing - Cause-Effect Graphing," Software Testing Services, 2005.
- [5] J. M. Cho, Y. H. Cho, and G. C. Kim, "A Corpus Formalization for Extracting the Syntactic Relations," Annual Conference on Human and Language Technology, pp.207-215, 1996.
- [6] G. Cho, J. Ku, and H. Chae, "The Impact of Jejueo Normalization and Dual Retrieval Paths on RAG Question

- Answering," International Journal of Internet, Broadcasting and Communication (IJIBC), Vol. 17, No. 4, pp. 109-116, 2025.
- [7] Chain-of-Thought Prompting, <https://www.promptingguide.ai/kr/techniques/cot> (accessed Nov. 20, 2025).
- [8] Japanese AI Prompting, <https://macrolingo.com/japanese-fukatsu-shunsuke-ai-prompting-methods/> (accessed Nov., 20, 2025).
- [9] W. Shin, "Utilizing Generative AI for Test Case Generation: Comparative Analysis and Guidelines," International Journal of Advanced Smart Convergence (IJASC), Vol. 13, No.4, pp. 145-154, 2024.
- [10] S. Yoon, A. Choi, C. Kim, S. Sohn, S. Oh, and M. Park, "A Study of Generative AI Trends and Applications," The Journal of the Convergence on Culture Technology (JCCT), Vol. 10, No. 4, pp.607-612, July 2024.
- [11] ChatGPT, <https://chatgpt.com/> (accessed Nov. 20, 2025).
- [12] Claude, <https://claude.ai/> (accessed Nov. 20, 2025).
- [13] Google Gemini, <https://gemini.google.com/> (accessed Nov. 20, 2025).
- [14] Microsoft Copilot, <https://copilot.microsoft.com/> (accessed Nov. 20, 2025).

※ 이 논문은 2020년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원 (No.2020-0-01373, 인공지능대학원지원(한양대학교))과 한양대학교 교내연구지원사업의 지원을 받아 수행된 연구임(HY-2024)