논문 2026-2-1 http://dx.doi.org/10.29056/jsf.2026.06.01

LLM 기반 요구사항-코드 의미 정합성 분석을 통한 소프트웨어 완성도 평가 모델

김유경*†

Requirement-Code Semantic Alignment based Software Completeness Assessment Model using LLM

Yukyong Kim *†

요 약

소프트웨어 완성도 평가는 소프트웨어 품질보증, 유지보수 검증, 외주 개발 검수 및 공공 소프트웨어 사업 평가에서 중요한 역할을 수행한다. 기존의 소프트웨어 평가 방법은 실제 구현이 사용자 요구사항을 의미적으로 충족하는지 평가하는 데에는 한계가 존재한다. 본 논문에서는 요구사항과 소스코드 간 의미 정합성을 기반으로 소프트웨어 완성도를 평가하는 LLM 기반 소프트웨어 감정 모델을 제안한다. 제안 모델은 자연어 요구사항과 소스코드의 의미 정보를 분석하고, 임베딩 기반 유사도 및 LLM 기반 추론을 통해 요구사항-구현 정합성을 평가한다. 또한 구현 누락 및 과잉 구현 기능을 탐지하고 설명가능한 평가 리포트를 자동 생성한다. 실험 결과, 제안 모델은 기존 키워드 기반 추적성 분석 기법 대비 우수한 정합성 분석 성능을 보였으며, 소프트웨어 완성도 평가의 설명 가능성과 자동화 수준을 향상시켰다. 제안 모델은 소프트웨어 감리, 유지보수 평가, 외주 개발 검증 및 공공 SW 품질 평가 분야에 활용될 수 있을 것으로 기대된다.

Abstract

Software completeness evaluation is important for software quality assurance, maintenance verification, outsourced software inspection, and public software project assessment. Existing approaches have limitations in evaluating whether software implementations semantically satisfy user requirements. This paper proposes an LLM-based software appraisal model that evaluates software completeness based on semantic alignment between requirements and source code. The proposed model extracts functional information from natural language requirements, analyzes semantic information from source code, and evaluates requirement-implementation alignment using embedding-based similarity and LLM-based reasoning. In addition, the model detects missing and over-implemented functionalities and automatically generates explainable evaluation reports. Experimental results show that the proposed model outperforms conventional keyword-based traceability approaches and improves the explainability and automation of software completeness evaluation. The proposed approach can be applied to software auditing, maintenance evaluation, outsourced software verification, and public software quality assessment.

한글키워드 : 대규모 언어모델, 소프트웨어 감정, 요구사항 분석, 의미 정합성, 소프트웨어 완성도

keywords : LLM, SW appraisal, requirements analysis, semantic alignment, SW completeness

* 숙명여자대학교 첨단공학부
† 교신저자: 김유경(email: ykim.be@sookmyung.ac.kr)

접수일자: 2026.06.01. 심사완료: 2026.06.04
게재확정: 2026.06.20.

1. 서 론

소프트웨어 완성도 평가는 소프트웨어 공학 분야에서 중요한 품질 관리 활동 중 하나이다. 특히 공공 소프트웨어 사업, 유지보수 계약, 외주 개발 검수 및 소프트웨어 감리에서는 요구사항 대비 구현 수준을 평가하는 과정이 필수적으로 요구된다. 기존의 소프트웨어 품질 평가는 코드 복잡도(Code Complexity), 테스트 커버리지(Test Coverage), 결함 밀도(Defect Density), 기능점수(Function Point Analysis; FPA) 등의 정량적 지표를 중심으로 수행되어 왔다[1][2]. 이러한 방법들은 정량적 품질 측정에는 효과적이지만, 실제 구현이 사용자 요구사항을 의미적으로 충족하는지 평가하기에는 한계가 존재한다.

요구사항 추적성(Requirement Traceability)은 요구사항과 설계, 코드, 테스트 간의 관계를 연결하고 추적하기 위한 핵심 연구 분야로 오랫동안 연구되어 왔다[3]. 기존의 요구사항 추적성 분석은 정보검색(Information Retrieval) 기반 접근법, TF-IDF(Vector Space Model), Latent Semantic Indexing(LSI), 규칙 기반 분석 등을 활용하였다[4][5]. 그러나 이러한 기법들은 주로 키워드 중심 매칭에 의존하기 때문에 문맥 기반 의미 이해가 부족하며, 요구사항과 코드 간 표현 차이로 인해 높은 오탐(False Positive) 및 미탐(False Negative)이 발생하는 문제가 존재한다.

최근 Transformer 기반 대규모 언어모델(Large Language Model; LLM)의 발전은 자연어와 소스코드에 대한 의미적 이해 수준을 크게 향상 시켰다[6]. GPT 계열 모델, CodeBERT, CodeLlama와 같은 최신 모델들은 자연어 요구사항 분석, 코드 생성, 코드 요약, 버그 탐지 및 문서 자동화 등 다양한 소프트웨어 공학 분야에 활용되고 있다[7][8]. 특히 자연어와 코드 간 의미 관계를 벡터 공간에서 표현할 수 있는 임베딩 기술의 발전은 요구사항과 구현 간 의미 정합성을 자동 분석할 가능성을 제시하고 있다.

LLM 기반 소프트웨어 분석 연구는 최근 빠르게 증가하고 있으나, 대부분 코드 생성, 코드 자동 완성, 코드 리뷰 및 버그 수정 분야에 집중되어 있다[9]. 반면 요구사항과 구현 간 의미 정합성을 기반으로 소프트웨어 완성도를 평가하고, 설명 가능한 감정(Appraisal) 결과를 제공하는 연구는 미흡한 실정이다.

소프트웨어 완성도 평가는 단순한 코드 품질 측정을 넘어 요구사항 대비 구현 수준, 기능 누락 여부, 과잉 구현 여부 및 품질 위험 요소를 종합적으로 분석해야 한다. 특히 공공 SW 사업 및 유지보수 검수 환경에서는 구현 완성도와 요구사항 준수 여부를 객관적으로 평가할 수 있는 자동화된 체계가 필요하다.

이에 본 논문에서는 요구사항과 소스코드 간 의미 정합성(Semantic Alignment)을 분석하여 소프트웨어 완성도를 평가하는 LLM 기반 모델을 제안한다. 제안 모델은 자연어 요구사항으로부터 기능 의미를 추출하고, 소스코드의 기능 의미를 분석한 뒤, 임베딩 기반 의미 유사도와 LLM 추론 기반 검증을 결합하여 요구사항-구현 정합성을 평가한다. 또한 구현 누락 기능, 부분 구현 기능 및 과잉 구현 기능을 탐지하고, 생성형 AI 기반 설명 가능 평가 리포트를 자동 생성한다.

본 논문의 구성은 다음과 같다. 2장에서는 요구사항 추적성, 소프트웨어 완성도 평가 및 LLM 기반 소프트웨어 분석 관련 연구를 소개한다. 3장에서는 제안하는 요구사항-코드 의미 정합성 기반 소프트웨어 감정 프레임워크와 완성도 평가 방법을 기술한다. 4장에서는 실험 환경 및 평가 결과를 분석한다. 마지막으로 5장에서는 결론 및 향후 연구 방향을 제시한다.

2. 관련 연구

2.1 요구사항 추적성 분석

요구사항 추적성(Requirement Traceability)은 요구사항과 설계 문서, 소스코드, 테스트케이스 간 관계를 연결하고 추적하는 활동을 의미한다. 요구사항 추적성은 소프트웨어 유지보수, 품질보증 및 감리 과정에서 핵심적인 역할을 수행하며, 요구사항 변경에 따른 영향 분석 및 구현 검증을 지원한다[3].

초기 요구사항 추적성 연구는 수작업 기반 매핑 방식에 의존하였다. 그러나 대규모 소프트웨어 프로젝트에서는 수작업 방식의 비용이 증가함에 따라 자동화된 추적성 분석 기법에 대한 연구가 활발히 진행되었다[10].

대표적인 자동 추적성 분석 기법으로는 정보검색 기반 접근법이 있다. Marcus와 Maletic [4]은 Latent Semantic Indexing(LSI)을 활용하여 문서와 코드 간 의미 기반 추적성을 분석하였으며, [5]에서 Hayes 등은 벡터 공간 모델을 활용한 요구사항 추적성 생성 기법을 제안하였다. 이후 TF-IDF, BM25, LSI 등 다양한 텍스트 기반 의미 분석 기법이 요구사항-코드 매핑 문제에 적용되었다. 이러한 기법들은 문서 간 단어 빈도와 통계적 특징을 활용하여 유사도를 계산한다. 그러나 키워드 중심 접근 방식은 요구사항과 코드 간 표현 차이(vocabulary mismatch)에 취약하며, 문맥 기반 의미 이해가 부족하다는 한계를 가진다.

최근에는 딥러닝 기반 요구사항 추적성 연구가 증가하고 있다. [11]에서 Guo 등은 Siamese Neural Network를 활용하여 요구사항과 코드 간 의미 유사도를 분석하였으며, [12]의 연구에서는 소프트웨어 요구사항 간의 추적성 링크를 LLM과 검색 증강 생성(RAG)을 활용해 자동으로 복구하는 새로운 방법론을 제안하였다.

기존 연구는 주로 “링크 존재 여부” 판단에 집중되어 있으며, 구현 완성도 및 의미 기반 소프트웨어 감정 관점의 분석은 충분히 수행되지 않았다.

2.2 소프트웨어 완성도 평가

소프트웨어 완성도 평가는 요구사항 대비 실제 구현 수준을 평가하는 활동으로, 소프트웨어 감리, 유지보수 검수 및 품질보증 과정에서 중요한 역할을 수행한다. 기존의 소프트웨어 완성도 평가는 주로 기능점수, 코드 복잡도(Complexity), 테스트 커버리지, 결함 밀도(Defect Density), 요구사항 커버리지(Requirement Coverage) 등과 같은 정량적 지표를 기반으로 수행된다[13][14].

기능점수분석은 소프트웨어 규모 및 생산성 측정 분야에서 널리 활용되고 있으며, 사용자 관점에서 기능 규모를 측정할 수 있다는 장점을 가진다. 그러나 실제 구현 품질이나 의미적 요구사항 충족 여부를 직접 평가하기에는 한계가 존재한다. McCabe의 순환 복잡도(Cyclomatic Complexity)는 코드 구조의 복잡성을 측정하는 대표적인 방법이며, 유지보수성과 결함 가능성을 평가하는 데 활용된다. 또한 테스트 커버리지는 테스트 수행 범위를 측정하여 품질 수준을 평가하는 데 사용된다.

그러나 이러한 기존 지표들은 대부분 코드 중심 정량 분석에 초점을 맞추고 있으며, 요구사항과 구현 간 의미적 일치 여부를 충분히 반영하지 못한다. 예를 들어 테스트 커버리지가 높더라도 요구사항 자체가 잘못 구현되었을 가능성이 존재한다.

최근에는 NLP 및 AI 기술을 활용하여 요구사항 기반 품질 평가를 수행하려는 연구가 증가하고 있다[15]. 하지만 대부분 제한적인 규칙 기반 분석 수준에 머물러 있으며, 설명 가능한 감정 리포트 생성 기능은 부족한 상황이다.

2.3 LLM 기반 소프트웨어 분석

최근 LLM의 발전은 소프트웨어 공학 분야에 큰 변화를 가져오고 있다. Transformer 구조를 기반으로 하는 GPT, BERT, CodeBERT 등의 모델은 자연어와 프로그래밍 코드 모두에 대한 의미 이해 능력을 보인다[6][8]. OpenAI의 GPT 계열 모델은 코드 생성, 코드 리뷰, 테스트케이스 생성 및 문서 자동화에 활용되고 있으며, GitHub Copilot과 같은 상용 시스템에도 적용되고 있다[7]. Feng 등이 제안한 CodeBERT는 자연어와 소스코드를 동시에 학습하여 코드 검색 및 코드 문서화 분야에서 우수한 성능을 보였다[8].

최근 연구에서는 LLM을 활용한 다양한 소프트웨어 공학 응용 연구가 활발히 진행되고 있다. 대표적으로 LLM은 코드 자동 생성, 코드 요약, 버그 탐지, 테스트케이스 생성, 코드 리뷰, 요구사항 분석 등의 분야에서 활용되고 있으며, 소프트웨어 개발 생산성과 품질 향상에 기여하고 있다. 특히 자연어와 소스코드 간 의미적 관계를 이해할 수 있는 능력을 기반으로 기존 규칙 기반 접근 방식의 한계를 극복할 수 있는 가능성을 보여주고 있다. [9]에서는 사전학습 언어모델을 활용한 코드 리뷰 자동화 기법을 제안하였으며, [16]의 연구에서는 Codex 기반 코드 생성 모델의 가능성을 제시하였다. 또한 최근에는 RAG 기반 코드 검색 및 요구사항 분석 연구도 증가하고 있다[17]. RAG 기반 접근법은 외부 문서 및 코드 저장소를 검색하여 보다 정확한 reasoning을 수행할 수 있다는 장점을 가진다. 그러나 현재까지의 연구는 코드 생성 및 코드 이해 중심으로 진행되고 있으며, 요구사항-코드 의미 정합성을 기반으로 소프트웨어 완성도를 평가하는 연구는 아직 부족한 상황이다. 또한 요구사항 대비 구현 누락 기능이나 과잉 구현 기능을 체계적으로 탐지하는 연구가 부족하다.

이에 본 연구에서는 이러한 한계를 해결하기 위해 LLM 기반 의미 정합성 분석을 활용한 소프트웨어 완성도 평가 모델을 제안한다. 제안 모델은 자연어 요구사항과 소스코드 간 의미 관계를 분석하여 구현 수준을 평가하고, 구현 누락 및 과잉 구현 여부를 탐지하며, 설명 가능한 감정 리포트를 자동 생성할 수 있도록 설계되었다.

3. 제안 프레임워크

3.1 프레임워크 개요

본 연구에서 제안하는 LLM 기반 요구사항-코드 의미 정합성 분석 프레임워크는 자연어 요구사항과 소스코드 간 의미 관계를 자동 분석하여 소프트웨어 완성도를 평가하기 위한 통합 구조로 설계되었다. 제안 프레임워크는 요구사항 명세로부터 기능 의미를 추출하고, 소스코드의 구현 의미를 분석한 뒤, 두 정보 간 의미 정합성을 평가하여 구현 수준을 정량·정성적으로 분석한다. 또한 생성형 AI 기반 설명가능 감정 리포트를 자동 생성함으로써 기존 정량 중심 평가 방식의 한계를 보완한다.

기존 요구사항 추적성 분석 시스템은 주로 키워드 매칭 또는 단순 문서 유사도 기반 접근 방식을 사용한다. 그러나 이러한 방식은 요구사항과 구현 코드 간 표현 차이(vocabulary mismatch)를 충분히 반영하지 못하며, 의미적 문맥을 이해하는 데 한계가 존재한다. 반면 본 연구의 프레임워크는 LLM 기반 임베딩 및 의미 추론(reasoning)을 활용하여 자연어 요구사항과 소스코드 간 의미 수준의 정합성을 분석한다.

제안된 프레임워크는 그림 1과 같이 다음의 5개 핵심 모듈로 구성된다.

- 요구사항 분석기(Requirement Analyzer)
- 코드 의미 추출기(Code Semantic Extractor)
- 의미 정합성 분석 엔진(Semantic Alignment

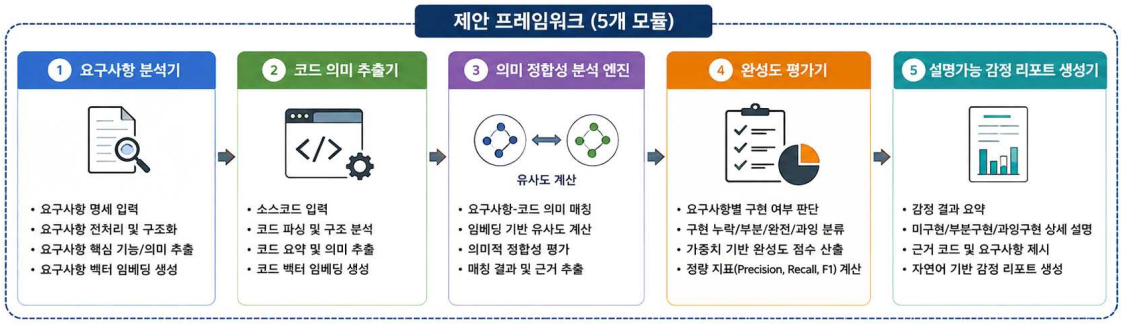

그림 1. 제안된 프레임워크 구조
Fig. 1. Architecture of the proposed framework

Engine)

- 완성도 평가기(Completeness Evaluator)
- 감정 리포트 생성기(Explainable Appraisal Report Generator)

각 모듈은 독립적으로 동작하면서도 상호 연계 구조를 가진다. 전체 시스템은 요구사항 입력 단계부터 최종 감정 리포트 생성 단계까지 순차적 파이프라인 구조로 구성된다.

5개 핵심 모듈을 통한 전체 평가 프로세스는 그림 2와 같다. 먼저 요구사항 분석기는 자연어 기반 요구사항 명세서를 입력받아 핵심 기능, 행위자(actor), 제약조건(constraint), 비기능 요구사항 등을 추출한다. 이후 LLM 기반 임베딩 모델을 활용하여 요구사항의 의미 벡터를 생성한다.

다음으로 코드 의미 추출기는 소스코드 및 함수 정보를 분석하여 코드의 기능 의미를 추출한다. 본 연구에서는 함수명, 클래스명, 주석(comment), API 호출 정보 및 코드 실행 흐름을 기반으로 코드 의미를 분석하며, 코드 기능에 대한 자연어 요약 정보와 임베딩 벡터를 생성한다.

의미 정합성 분석 엔진은 요구사항 임베딩과 코드 임베딩 간 유사도를 계산하여 의미 정합성을 평가한다. 또한 단순 벡터 유사도뿐 아니라 LLM 기반 추론 기법을 활용하여 요구사항과 코드 간 기능적 일치 여부를 검증한다. 이를 통해 단순 키워드 일치가 아닌 의미 기반 요구사항-구현 관계를 분석할 수 있다.

완성도 평가기는 요구사항별 구현 여부를 분석하여 전체 소프트웨어 완성도를 계산한다. 본 연구에서는 요구사항 커버리지, 의미 정합성 점수, 구현 누락 등을 종합적으로 고려하여 완성도 점수를 산출한다.

마지막으로 감정 리포트 생성기는 분석 결과를 기반으로 자연어 기반 감정 리포트를 생성한다. 기존 정량 중심 품질 평가 시스템과 달리, 본 시스템은 “어떤 요구사항이 구현되었는가”, “어떤 기능이 누락되었는가”, “어떤 기능이 과잉 구

그림 2. 평가 프로세스
Fig. 2. Assessment process

현되었는가" 등을 설명 가능한 형태로 제시할 수 있다. 이를 통해 감리자, 프로젝트 관리자 및 유지보수 담당자의 이해도를 향상시킬 수 있다.

또한 본 연구에서는 제안 프레임워크의 실험 및 검증을 위해 Streamlit 기반 실험 플랫폼을 구현하였다. 실험 플랫폼은 요구사항과 코드 요약 정보를 입력받아 의미 정합성 분석 결과, Precision/Recall/F1-score, 완성도 점수 및 감정 리포트를 시각적으로 제공한다. 특히 유사도 임계값(threshold)을 조절하여 의미 정합성 변화에 따른 성능 차이를 분석할 수 있도록 구성하였다.

3.2 요구사항-코드 정합성 평가 과정

제안 프레임워크의 핵심 구성 요소인 5개 모듈들은 자연어 요구사항과 소스코드 간 의미 관계를 분석하여 요구사항 대비 구현 수준을 평가하기 위한 핵심 기능을 수행한다. 기존 요구사항 추적성 분석 시스템은 주로 키워드 매칭 또는 통계 기반 문서 유사도 분석에 의존하였다. 그러나 이러한 방식은 요구사항과 코드 간 표현 차이(vocabulary mismatch) 및 문맥 정보 부족 문제를 충분히 해결하지 못하였다. 반면 본 연구에서는 LLM 기반 임베딩 및 의미 추론 기법을 활용하여 요구사항과 구현 코드 간 의미 수준의 정합성을 분석한다.

요구사항 분석기는 자연어 기반 요구사항 명세서를 입력받아 핵심 기능 의미를 추출하는 역할을 수행한다. 요구사항 문장은 일반적으로 비정형 자연어 형태로 작성되기 때문에, 기능 추출 및 구조화 과정이 필요하다. 본 연구에서는 LLM 기반 자연어 분석 기법을 활용하여 행위자(Actor), 기능(Function), 객체(Object), 제약조건(Constraint), 비기능 요구사항(Non-functional Requirement) 정보를 추출한다. 예를 들어, "사용자는 이메일 인증을 통해 비밀번호를 재설정할 수 있어야 한다."와 같은 요구사항이 존재한다고 가정하면, 요구사항 분석기는 이를 표 1과 같이 구조화한다.

표 1. 구조화된 추출 정보
Table 1. Extracted information

요소	추출 결과
Actor	사용자
Function	비밀번호 재설정
Constraint	이메일 인증

추출된 요구사항 정보는 이후 의미 정합성 분석을 위해 문장 임베딩 형태로 변환된다. 본 연구에서는 SentenceTransformer 기반 임베딩 모델을 활용하여 요구사항 의미 벡터를 생성하였다. 요구사항 임베딩은 다음과 같이 표현된다.

$$R_i = Embedding(Requirement_i)$$

여기에서 R_i는 i번째 요구사항의 의미임베딩 벡터를 나타낸다. 또한 요구사항 분석기는 기능 요구사항뿐 아니라 성능, 보안, 인증 등의 비기능 요구사항도 함께 분석할 수 있도록 설계하였다. 이를 통해 기존 기능 중심 평가 방식의 한계를 보완할 수 있다.

코드 의미 추출기는 소스코드를 분석하여 코드의 기능 의미를 추출하는 역할을 수행한다. 일반적인 정적 분석 도구는 코드 복잡도나 구조 중심 정보를 제공하지만, 실제 코드가 수행하는 기능 의미를 충분히 설명하지 못한다. 본 연구에서는 함수명, 클래스명, 주석, API 호출 정보 및 실행 흐름을 기반으로 코드 의미를 분석한다. 예를 들어 다음과 같은 코드가 존재한다고 가정한다.

```
def reset_password(email) :
    send_auth_code(email)
```

코드 의미 추출기는 해당 코드를 다음과 같이 자연어 형태로 요약한다. "이 함수는 이메일 인증 기반 비밀번호 재설정 기능을 수행한다." 이후 생성된 코드 의미 설명은 임베딩 벡터로 변환

된다.

$$C_j = Embedding(Code_j)$$

여기서 C_j는 j번째 코드의 의미 임베딩 벡터를 의미한다. 코드 의미 추출기는 단순 함수 수준 분석뿐 아니라, API 호출 패턴, 데이터 처리 흐름, 인증 및 권한 처리, 예외 처리 로직, 데이터베이스 접근 패턴 등의 요소도 함께 고려한다. 이를 통해 코드의 실제 기능 의미를 보다 정확하게 추출할 수 있다. 또한 본 연구에서는 LLM 기반 코드 요약 기법을 활용하여 코드 기능 설명을 자동 생성하였다. 이는 요구사항과 코드 간 표현 차이를 줄이는 데 중요한 역할을 수행한다.

의미 정합성 분석 엔진은 요구사항 임베딩과 코드 임베딩 간 의미 유사도를 계산하여 요구사항-구현 정합성을 평가하는 핵심 모듈이다. 기존 TF-IDF 기반 분석은 단어 빈도 중심 유사도 계산에 의존하기 때문에 의미적 문맥을 충분히 반영하지 못한다. 반면 본 연구에서는 임베딩 기반 의미 유사도와 LLM 기반 추론(reasoning)을 결합하여 보다 정교한 정합성 분석을 수행한다. 요구사항 벡터와 코드 벡터 간 의미 유사도는 코사인 유사도(Cosine Similarity)를 사용하여 계산하였다.

$$Similarity(R_i, C_j) = \frac{R_i \cdot C_j}{\| R_i \| \| C_j \|}$$

여기에서 R_i는 요구사항 임베딩 벡터, C_j는 코드 임베딩 벡터를 의미한다. 코사인 유사도는 두 벡터의 방향 유사성을 측정하는 방법으로, 0-1 범위로 계산되며, 값이 높을수록 의미 정합성이 높음을 의미한다. 본 연구에서는 threshold 기반 매칭 방식을 사용하였다. 즉, 의미 유사도가 특정 임계값 이상인 경우 요구사항과 코드가 의미적으로 매칭된 것으로 판단한다.

$$M(R_i, C_j) = \begin{cases} 1, \text{if } Similarity(R_i, C_j) \geq Threshold \\ 0, \text{otherwise} \end{cases}$$

또한 단순 벡터 유사도 계산만으로는 의미적 구현 여부를 완전히 판단하기 어렵기 때문에, 본 연구에서는 LLM 기반 추론 검증 과정을 추가하였다. 예를 들어, 요구사항이 “시스템은 비밀번호를 암호화하여 저장해야 한다.”라고 가정해보자. 단순 키워드 기반 분석은 “저장” 또는 “비밀번호” 키워드만으로도 높은 유사도를 계산할 수 있다. 그러나 본 연구의 의미 정합성 분석 엔진은 실제 코드에 SHA256, AES 등의 암호화 처리 로직이 존재하는지를 추가적으로 검증한다. 이를 통해 요구사항 대비 구현 수준을 분석할 수 있으며, 구현 상태를 완전 구현, 부분 구현, 미구현으로 구분하여 평가할 수 있다. 최종적으로 의미 정합성 분석 결과는 완성도 평가 모듈로 전달되며, 요구사항별 구현 수준 평가 및 감정 리포트 생성에 활용된다.

3.3 완성도 산출

본 연구에서는 요구사항 충족 수준을 기반으로 완성도 점수를 산출한다. 전체 완성도는 다음과 같이 계산한다.

$$Completeness = \frac{\sum_{i=1}^{n} ImScore_i}{TotalReq} \times 100$$

여기에서 $ImScore_i$는 구현수준에 대한 가중치를 사용한다. 가중치는 다음 표 2와 같다.

표 2. 구현 수준에 대한 가중치
Table 2. Weights for implementation levels

구현수준	가중치
완전 구현	1.0
부분 구현	0.5
미구현	0

예를 들어, 완전 구현 12개, 부분 구현 2개, 미구현 1개 인 경우 완성도는 다음과 같이 계산된

다.

$$Completeness = \frac{12 + 2 \times 0.5}{15} \times 100 = 86.7$$

본 연구에서는 LLM을 활용하여 자연어 기반 설명가능 감정 리포트를 생성한다. 생성되는 리포트는 요구사항 구현 여부, 구현 누락 기능, 부분 구현 기능, 품질 위험 요소 등을 포함한다. 예를 들어 다음과 같은 결과를 생성할 수 있다.

- 사용자 인증 요구사항은 구현 코드와 높은 의미 정합성을 보이며 정상적으로 구현된 것으로 확인되었다.
- 비밀번호 암호화 기능은 존재하나 비밀번호 강도 검증 기능이 누락되어 부분 구현 상태로 평가되었다.
- 응답시간 3초 이내 요구사항은 구현 근거를 확인할 수 없어 미구현으로 평가되었다.

이와 같은 설명가능 리포트는 프로젝트 관리자, 감리자 및 유지보수 담당자가 결과를 직관적으로 이해할 수 있도록 지원한다.

4. 실험 및 평가

4.1 실험 환경

본 연구에서는 제안한 LLM 기반 요구사항-코드 의미 정합성 분석 모델의 성능을 검증하기 위하여 실험용 평가 플랫폼을 구현하였다.

실험 플랫폼은 Python과 Streamlit을 기반으로 구현되었으며, 요구사항 명세서와 코드 기능 설명을 입력받아 의미 정합성 분석, 완성도 평가 및 감정 리포트 생성을 수행하도록 설계하였다. 실험 환경은 표 3과 같다.

실험 데이터셋은 요구사항 명세, 코드 기능 요약, 그리고 정답 매핑 데이터로 구성하였다. 실험 데이터는 사용자 관리 시스템, 게시판 시스템 및 인증 시스템을 가정하여 구축하였다.

표 3. 실험 환경
Table 3. Experimental environment

항목	내용
운영체제	Windows 11
개발언어	Python 3.11
프레임워크	Streamlit 1.35
임베딩 모델	SentenceTransformer (all-MiniLM-L6-v2)
유사도 계산	Cosine Similarity
데이터 처리	Pandas
평가 라이브러리	Scikit-learn

데이터셋은 실제 소프트웨어 프로젝트에서 발생할 수 있는 다양한 상황을 반영하기 위해 다음과 같은 유형의 요구사항을 포함하였다.

- 사용자 인증 기능
- 게시판 기능
- 관리자 기능
- 보안 요구사항
- 성능 요구사항
- 사용자 인터페이스 요구사항

총 15개의 요구사항과 20개의 코드 기능 설명을 구성하였다. 요구사항 중 일부는 완전히 구현된 상태로 설정하였으며, 일부는 부분 구현 또는 미구현 상태로 구성하였다. 또한 실제 프로젝트 환경을 모사하기 위해 요구사항에 존재하지 않는 기능, 유사 기능을 수행하는 코드, 관리자 전용 기능, 실시간 알림 기능과 같은 Noise 데이터를 포함하였다.

Ground Truth 데이터는 전문가 검토를 통해 요구사항과 코드 간 실제 매핑 관계를 정의하였다. 총 15개 요구사항 중 14개는 대응되는 구현 기능이 존재하며, 1개 요구사항은 의도적으로 미구현 상태로 설정하였다. 그림 3은 요구사항-코드 의미 정합성 분석 결과, Precision, Recall, F1 score, 소프트웨어 완성도 점수 및 설명가능 감정

리포트를 제공하는 실험 플랫폼의 스크린샷이다.

그림 3. 실험플랫폼 스크린샷
Fig. 3. Screenshot of the experimental platform

4.2 실험 방법 및 결과

본 연구에서는 제안 모델의 의미 정합성 평가 성능을 검증하기 위해 의미정합성 분석 성능 평가, 유사도 임계값 변화 평가, 소프트웨어 완성도 평가를 수행하였다.

의미 정합성 분석 성능 평가에서는 요구사항과 코드 간 의미 정합성 분석 정확도를 기존의 키워드 매칭방법과 비교하였다. 평가 지표는 Precision, Recall 및 F1-score를 사용하였다. 실험 결과는 다음 표 4와 같다. 키워드 기반 방법은 단순 용어 일치에 의존하기 때문에 의미적 문맥 차이를 충분히 반영하지 못하였다. 반면 제안 모델은 임베딩 기반 의미 분석과 문맥 이해를 통해 높은 정합성 평가 성능을 보였다. 특히 요구사항과 코드 간 표현 방식이 서로 다른 경우에도 의미적 유사성을 효과적으로 탐지할 수 있었다.

표 4. 의미정합성 분석 성능 비교
Table 4. Comparison of performance

방법	키워드 매칭	제안 모델
Precision	0.61	0.88
Recall	0.55	0.84
F1 score	0.58	0.86

이와 함께 유사도 임계값 변화 실험은 의미 유사도 임계값(threshold)에 따른 성능 변화를 분석하였다. 표 5에서 보여지는 것과 같이 실험 결과 임계값이 증가할수록 Precision은 향상되었으나 Recall은 감소하는 경향을 보였다. 이는 높은 임계값이 보다 엄격한 정합성 판단을 수행하기 때문이다.

표 5. 임계값 민감도 분석
Table 5. Threshold sensitivity analysis

Threshold	Precision	Recall	F1-score
0.4	0.72	0.95	0.82
0.5	0.81	0.91	0.86
0.6	0.88	0.84	0.86
0.7	0.91	0.72	0.80
0.8	0.95	0.60	0.73

Threshold 0.4에서는 대부분의 요구사항이 매칭되어 Recall은 높았지만 오탐이 증가하였다. 반면 Threshold 0.8에서는 엄격한 매칭 기준으로 인해 Precision은 높아졌으나 일부 실제 관계를 탐지하지 못하여 Recall이 감소하였다. 그림 4와 같이 전체적으로 Threshold 0.6 환경에서 가장 균형적인 성능을 보였다. 마지막으로 요구사항 충족 수준을 기반으로 소프트웨어 완성도를 평가하였다. 총 15개 요구사항 중 12개 요구사항이 완전 구현 상태로 확인되었으며, 2개 요구사항은

부분 구현, 1개 요구사항은 미구현 상태로 분석되었다. 부분 구현 요구사항에 대해 표 2에서 제시한 바와 같이 0.5 가중치를 적용한 결과 전체 완성도는 86.7%로 산출되었다.

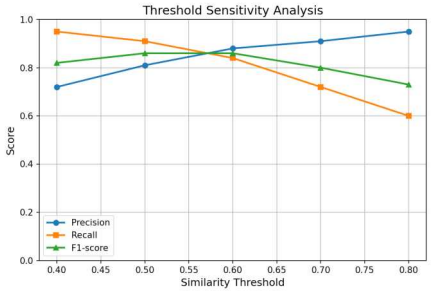

그림 4. 임계값 분석 결과
Fig. 4. Results of threshold analysis

또한 제안 모델은 단순 수치 결과뿐 아니라 자연어 기반 감정 리포트를 생성한다. 예를 들어 다음과 같은 결과를 제공하였다.

- 사용자 인증 기능은 요구사항과 높은 의미 정합성을 보이며 정상적으로 구현된 것으로 확인되었다.
- 비밀번호 암호화 기능은 구현되어 있으나 비밀번호 강도 검증 기능이 누락되어 부분 구현 상태로 평가되었다.
- 시스템 응답시간 3초 이내 요구사항은 구현 근거를 확인할 수 없어 미구현으로 평가되었다.

이러한 결과는 기존 정량 중심 평가 방식보다 감리자 및 프로젝트 관리자에게 높은 설명 가능성을 제공하였다.

실험 결과 제안 모델은 기존 키워드 매칭 방법 대비 높은 의미 정합성 분석 성능을 보였다. 특히 요구사항과 코드 간 표현 방식이 상이한 경우에도 의미 기반 매칭이 가능함을 확인하였다. 또한 완성도 평가 기능을 통해 구현 누락, 부분 구현 및 과잉 구현 기능을 식별할 수 있었으며, 설명가능 감정 리포트를 통해 평가 결과의 해석 가능성을 향상시켰다. 이는 기존 코드 중심 품질 평가 기법이 제공하지 못했던 요구사항 중심 완성도 평가 체계를 제공할 수 있음을 의미한다. 따라서 제안 모델은 소프트웨어 감리, 유지보수 검증, 외주 개발 검수 및 공공 SW 품질 평가 분야에 효과적으로 활용될 수 있을 것으로 판단된다. 다만 본 연구는 비교적 소규모 데이터셋 기반으로 수행되었으며, 실제 산업 환경 수준의 대규모 프로젝트 데이터 적용은 향후 연구 과제로 남아 있다.

5. 결 론

본 연구에서는 요구사항과 소스코드 간 의미 정합성(Semantic Alignment)을 기반으로 소프트웨어 완성도를 평가하는 LLM 기반 소프트웨어 완성도 평가 모델을 제안하였다. 기존의 소프트웨어 품질 평가 및 완성도 분석 방법은 기능점수, 코드 복잡도, 테스트 커버리지 등 정량적 지표에 주로 의존하고 있으며, 요구사항이 실제 구현에 얼마나 충실하게 반영되었는지를 평가하는 데에는 한계가 존재한다.

이를 해결하기 위해 본 연구에서는 자연어 요구사항과 소스코드 간 의미 관계를 분석할 수 있는 LLM 기반 의미 정합성 분석 프레임워크를 설계하였다. 제안 모델은 요구사항 분석기, 코드 의미 추출기, 의미 정합성 분석 엔진, 완성도 평가기, 설명가능 감정 리포트 생성기로 구성되며, 요구사항과 구현 코드 간 의미 유사도를 계산하고 LLM 기반 추론을 통해 구현 수준을 평가하도록 설계되었다. 또한 요구사항 충족 여부를 기반으로 구현 상태를 완전 구현, 부분 구현 및 미구현으로 분류하고, 구현 누락 및 과잉 구현 기능을 탐지하여 소프트웨어 완성도를 산출할 수 있도록 하였다.

실험에서는 Streamlit 기반 평가 플랫폼을 구현하고 요구사항 데이터셋과 코드 기능 설명 데이터셋을 활용하여 제안 모델의 성능을 검증하였다. 실험 결과, 제안 모델은 기존 Keyword Matching 분석 기법 대비 높은 평가지표를 나타냈고, 특히 요구사항과 코드 간 표현 방식이 상이한 경우에도 높은 의미 정합성 분석 성능을 보였다. 또한 Threshold 변화 실험을 통해 제안 모델이 요구사항-구현 관계를 안정적으로 탐지할 수 있음을 확인하였다. 아울러 제안 모델은 단순히 요구사항과 구현 간 매칭 여부를 판단하는 수준을 넘어 구현 누락 기능, 부분 구현 기능을 식별할 수 있었으며, 이를 기반으로 자연어 형태의 설명가능 감정 리포트를 생성하였다. 이러한 결과는 기존 정량 중심 품질 평가 방식의 한계를 보완하고, 감리자 및 프로젝트 관리자가 소프트웨어의 구현 상태를 보다 직관적으로 이해할 수 있도록 지원할 수 있음을 보여준다.

그러나 본 연구에서 사용한 실험 데이터셋은 연구 목적에 맞게 구축된 제한적인 규모이므로 대규모 산업 프로젝트에 대한 검증이 추가적으로 필요하다. 또한 현재 구현된 의미 정합성 분석은 주로 기능 요구사항을 대상으로 수행되므로 성능, 보안, 사용성 등 비기능 요구사항에 대한 평가 기능은 제한적이다. 향후 연구에서는 이러한 한계를 보완하여 요구사항 명세서, 설계 문서, 소스코드 및 테스트케이스를 통합적으로 분석할 수 있는 소프트웨어 감정 체계를 구축할 예정이다. 이를 통해 요구사항 중심의 자동화된 소프트웨어 완성도 평가 및 감정 체계를 구축하고, 소프트웨어 품질 관리의 효율성과 객관성을 향상시키는 데 기여하고자 한다.

참 고 문 헌

[1] R. Shatnawi, W. Li, “The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process”, Journal of Systems and Software, vol. 81, no. 11, pp. 1868-1882, 2007. https://doi.org/10.1016/j.jss.2007.12.794

[2] S. M. Shah, M. Morisio, M. Torchiano, “The impact of process maturity on defect density”, In Proceedings of the international symposium on Empirical software engineering and measurement, pp.315 - 318, 2012. https://doi.org/10.1145/2372251.2372308

[3] O. Gotel, A. Finkelstein, “An Analysis of the Requirements Traceability Problem,” Proceedings of International Conference on Requirements Engineering, pp. 94 - 101, 1994. https://doi.org/10.1109/ICRE.1994.292398

[4] A. Marcus, J. Maletic, “Recovering Documentation to Source Code Traceability Links using Latent Semantic Indexing,” Proceedings of the International Conference on Software Engineering, pp. 125 - 135, 2003.

[5] J. H. Hayes, A. Dekhtyar, S. Sundaram, “Advancing Candidate Link Generation for Requirements Tracing,” IEEE Transactions on Software Engineering, Vol. 32, No. 1, pp. 4 - 19, 2006. https://doi.org/10.1109/TSE.2006.3

[6] A. Vaswani et al., “Attention Is All You Need,” In Proceedings of the International Conference on Neural Information Processing Systems, pp.6000-6010, 2017. ISBN: 9781510860964

[7] OpenAI, “GPT-4 Technical Report,” 2023. https://doi.org/10.48550/arXiv.2303.08774

[8] Z. Feng et al., “CodeBERT: A Pre-Trained Model for Programming and Natural Languages,” Findings of the Association for Computational Linguistics: EMNLP 2020, pp.1536-1547, 2020. https://doi.org/10.18653/v1/2020.findings-emnlp.139

[9] R. Tufano et al., “Using Pre-Trained Models to Boost Code Review Automation,” In Proceedings of the International Conference on Software Engineering, pp.2291 – 2302, 2022. https://doi.org/10.1145/3510003.3510621
[10] J. Cleland-Huang, O. Gotel, and A. Zisman, “Software and Systems Traceability”, Springer, 2012. ISBN:978-1-4471-2238-8
[11] J. Guo et al., “Semantically Enhanced Software Traceability using Deep Learning Techniques,” Proceedings of ICSE, 2017. https://doi.org/10.1109/ICSE.2017.9
[12] T. Hey et al., “Requirements Traceability Link Recovery via Retrieval-Augmented Generation,” Proceedings of the International Conference on Requirements Engineering: Foundation for Software Quality, pp.381-397, 2025. https://doi.org/10.1007/978-3-031-88531-0_27
[13] T. O. Hovorushchenko, “Methodology of Evaluating the Sufficiency of Information for Software Quality Assessment According to ISO 25010”, Journal of Information and Organizational Sciences, vol. 42, no. 1, p.63-85, 2018. https://doi.org/10.31341/jios.42.1.4
[14] D. Kim, “Software Completeness Evaluation based on ISO/IEC9241.10”, Journal of Software Assessment and Valuation, vol. 15, no. 2, pp.9-16, 2019. https://doi.org/10.29056/jsav.2019.12.02
[15] T. Menzies et al., “Data Mining Static Code Attributes to Learn Defect Predictors,” IEEE Transactions on Software Engineering, Vol. 33, No. 1, pp. 2 - 13, 2007. https://doi.org/10.1109/TSE.2007.256941
[16] M. Chen et al., “Evaluating Large Language Models Trained on Code,” arXiv preprint 2021. https://doi.org/10.48550/arXiv.2107.03374
[17] P. Lewis et al., “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” In Proceedings of the International Conference on Neural Information Processing Systems, Article No. 793, pp. 9459 – 9474, 2020.

저 자 소 개

김유경(Yukyong Kim)

2005.9-2006.8 UC Davis, Post-doc.
2006.9-2013.9 한양대학교 컴퓨터공학과 연구교수
2018.3-현재 숙명여자대학교 첨단공학부 교수
<주관심분야> 웹서비스 QoS 평가, SOA 기반 IoT 신뢰 평가, 소프트웨어 품질 평가