논문 2025-2-1 http://dx.doi.org/10.29056/jsav.2025.06.01

NLP 기반 요구사항 추출과 소스코드 매핑을 통한 소프트웨어 완성도 평가 프레임워크

김유경*†

A Framework for Software Completeness Evaluation Using NLP-Based Requirement Extraction and Code Mapping

Yukyong Kim*†

요 약

본 논문은 소프트웨어 개발 계약 또는 외주 프로젝트에서 발생하는 완성도 분쟁의 핵심 원인인 "요구사항 구현 여부의 불명확성" 문제를 해결하고자 한다. 제안된 프레임워크는 자연어 처리 기술(NLP)을 활용하여 요구사항을 추출하고, 이를 딥러닝 기반의 소스코드 분석 기법을 결합하여 구현된 코드가 계약 상 요구사항을 얼마나 충족하고 있는지를 정량적으로 판단한다. 이를 통해 기존의 감정인의 주관적 판단에 의존하던 소프트웨어 완성도 감정에서 객관성과 신뢰성을 부여할 수 있는 지원 도구로 활용할 수 있다. 정량적·정성적 기준을 모두 충족시키면서, 감정 절차의 시간 단축 및 비용 절감이 가능할 것으로 기대한다. 사례 연구를 통해 이 방법론의 유효성을 입증하고, 향후 감정 지원 도구로서 활용 가능성을 제시한다.

Abstract

Disputes over Disputes over software completeness frequently arise in development contracts and outsourced projects due to the ambiguity surrounding the implementation status of contractual requirements. This paper proposes a novel framework to address this issue by introducing an objective and automated approach to software completeness evaluation. The framework utilizes NLP techniques to extract requirements from contract documents and matches them with implemented source code using deep learning-based analysis. By quantitatively assessing the degree to which the implemented code fulfills the specified requirements, the framework reduces reliance on the subjective judgments of human evaluators. It supports both quantitative and qualitative evaluation criteria, offering improvements in assessment reliability, cost-efficiency, and time-effectiveness. A case study is presented to validate the framework's effectiveness, demonstrate its applicability as a practical tool for appraisal, quality assurance, and development monitoring.

한글키워드 : 소프트웨어 완성도, 자연어 처리, 요구사항 추출, 딥러닝 기반 코드분석

keywords : sw completeness, NLP, requirement extraction, deep-learning based code analysis

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

접수일자: 2025.05.10. 심사완료: 2025.05.17.
게재확정: 2025.06.20.

1. 서 론

소프트웨어 개발 계약 또는 외주 프로젝트에서 요구사항 이행 여부와 개발 결과물의 완성도를 둘러싼 분쟁이 빈번히 발생하고 있다. 계약 기반 개발 프로젝트에서 발생하는 72%의 분쟁이 요구사항 해석의 차이에서 기인한다[1]. 이러한 분쟁의 법적 해결 절차에서 완성도 감정은 중요한 기술적 판단 근거로 작용한다. 완성도 감정은 계약서에 제시된 요구사항을 기준으로, 실제 개발된 소프트웨어가 해당 요구사항을 어느 정도 충족하고 있는지를 기술적·객관적으로 평가하는 행위를 말한다. 그러나 요구사항이 소스코드에 제대로 구현되었는지 확인하는 것은 매우 어려운 작업이다. 이 과정은 주로 감정인이나 전문가에 의해 수행되며, 분쟁 해결, 납품 검수, 품질 인증 등을 목적으로 한다. 소프트웨어 감정 평가에서 '완성도'는 매우 중요한 기준이지만, 전통적인 감정 방식은 평가자의 주관에 따라 편차가 발생할 수 있으며, 이는 평가자 간 일치도가 58%에 그치는 것으로 나타난다. 특히 계약서 기반 감정에서 요구사항 구현 여부를 명확히 판단하기 어려울 뿐만 아니라 시간과 비용의 문제가 발생하게 된다[2].

이에 본 논문에서는 자연어 요구사항을 자동 분석하여 기능/비기능 요구사항을 구조화된 형태로 정리하고, 딥러닝 기반의 코드 분석 모델을 통해 실제 구현된 소스코드와 매핑함으로써 요구사항 충족 여부를 자동 판단하는 프레임워크를 제안한다. 본 연구의 목적은 요구사항과 소스코드 간의 매핑을 통해 소프트웨어의 완성도를 평가하고, 이 과정을 자동화하여 소프트웨어 완성도 감정 과정에서 활용할 수 있는 지원 방법을 제시하는 것이다.

본 논문은 다음과 같이 구성된다. 2장에서는 관련 연구를 기술하고, 3장에서는 제안된 프레임워크 아키텍처의 개요와 요구사항 추출, 소스코드 분석, 완성도 평가 과정에 대해 설명한다. 4장에서는 사례연구를 통해 제안된 방법론의 활용 과정을 다룬다. 마지막으로 5장에서는 결론과 향후 연구 방향을 논의한다.

2. 관련 연구

자연어 요구사항을 추출하는 연구는 최근 자연어 처리(NLP) 기술 발전에 따라 활발히 진행되고 있다. 특히, 소프트웨어 요구사항 공학(SRE) 분야에서는 요구사항의 명확성, 일관성, 추적 가능성을 확보하기 위해 다양한 접근법이 제안되고 있다. Ghosh 등은 자연어로 작성된 요구사항을 분석 가능한 형식 모델로 변환하는 프레임워크인 ARSENAL을 제안하였다. 이 시스템은 요구사항을 형식 논리로 변환하여 일관성과 구현 가능성을 자동으로 분석할 수 있도록 지원한다[3]. BOSCH의 자동차 도메인 요구사항을 대상으로 한 연구에서는 자연어 요구사항을 형식 명세로 변환하는 NLP 기반 파이프라인인 Req2Spec을 개발하였다. 이 시스템은 222개의 요구사항 중 71%를 정확하게 형식화하는 성과를 보였다[4]. Veizaga 등은 자연어 요구사항의 품질 문제(예: 모호성)를 자동으로 감지하고 개선을 위한 권고를 제공하는 도구인 Paska를 개발하였다. 이 도구는 금융 도메인의 13개 시스템에서 2,725개의 요구사항을 분석하여 89%의 정밀도와 재현율을 달성하였다[5].

이와 함께 요구사항 간 관계 및 구조 정보 추출하는 연구도 진행되었다. Motger와 Franch는 요구사항 간의 추적 가능성을 확보하기 위해 자연어 기반 관계 추출 기법을 체계적으로 정리하였다. 이들은 구문 기반 및 의미 기반 접근법, 정보 검색 기반 및 기계 학습 기반 방법 등을 분석

했다[6]. Frattini 등은 요구사항 문서에서 인과 관계를 자동으로 추출하는 시스템을 개발했다. 이 시스템은 4,457개의 문장을 분석하여 약 12.5%의 인과 관계 문장을 식별하고, 평균 48.6%의 인과 그래프를 자동 생성하는 성과를 보였다[7].

Necula 등은 1991년부터 2023년까지의 NLP와 소프트웨어 요구사항 공학의 통합 연구를 체계적으로 분석하였다. 이들은 NLP와 AI 기술이 요구사항의 명확성, 품질 향상, 자동화 등에 기여하고 있으며, 다국어 처리, 대규모 언어 모델(LLM)의 적용, 자동화 수준 향상 등이 향후 주요 연구 방향임을 제시하고 있다[8]

소스코드에서 요구사항이 얼마나 반영되었는지를 평가하는 연구는 주로 정적 분석 도구를 이용한다. 코드의 함수, 변수, 메소드 이름 등을 분석하여 해당 코드가 구현하는 기능을 파악한다. 또한 주석 분석 및 API 호출 분석을 통해 코드의 의도를 해석하는 방법이 사용된다. 요구사항-소스코드 간 추적성 확보를 위한 자동화에 관한 연구로는 [9]에서 제안한 YamenTrace가 있다. 이것은 LSI(Latent Semantic Indexing)와 FCA(Formal Concept Analysis)를 활용하여 객체지향 소프트웨어에서 요구사항과 소스코드 간의 추적 가능성을 자동으로 복원하고 시각화하는 도구이다. 코드 식별자, 주석, 관계 등을 활용하여 요구사항과 코드 간의 의미적 유사성을 분석했다. Chen 등은 소스코드와 문서 간의 추적 링크를 검색하고 시각화하는 시스템인 DCTracVis를 개발하였다. 이 시스템은 정보 검색 기법을 활용하여 요구사항과 코드 간의 연관성을 분석한다[10]. 최근 연구에서는 신경망 모델을 활용하여 요구사항과 소스코드 간의 추적 링크를 구축하는 방법이 제안되었다. 예를 들어, 특정 연구에서는 요구사항과 코드의 의미적 정보를 벡터로 임베딩한 후, 이들 간의 코사인 유사도를 계산하여 추적 링크를 생성하는 방식을 소개하였다[11].

이러한 연구들은 자연어로 작성된 요구사항의 자동 추출과 분석을 통해 소프트웨어 개발 과정의 효율성과 정확성을 향상 시키는 데 기여하고 있다. 또한 소프트웨어 개발 과정에서 요구사항과 소스코드 간의 연계를 강화하고, 완성도 평가를 자동화 한다. 기존 연구에서는 요구사항이 얼마나 구현되었는지를 평가하는 다양한 지표를 제시하고 있으며, 이를 소프트웨어 품질 평가의 중요한 도구로 활용한다.

소프트웨어 완성도는 소프트웨어가 설정된 요구사항을 얼마나 충족하는지를 나타내는 지표이다. 소프트웨어 완성도에 대한 연구는 ISO/IEC 25010과 같은 국제표준을 기반으로 소프트웨어 품질 특성을 체계적으로 분석하고 평가하는데 중점을 두고 있다. Hovorushchenko는 ISO/IEC 25010 표준에 따라 소프트웨어 품질을 평가하기 위해 형식화된 온톨로지 모델을 개발하였다. 이 연구는 소프트웨어 요구사항 명세서(SRS)의 정보 충분성을 평가하여 품질 평가의 신뢰성을 향상시키는 방법을 제안하고 있다[12]. 또한 [13]의 연구에서는 ISO/IEC 9241.10 표준을 기반으로 소프트웨어의 사용성과 기능적 완성도를 평가하는 방법론을 제안하였다. 이 연구는 기존의 기능 구현 여부 중심의 평가에서 벗어나, 사용성 측면을 포함한 보다 포괄적인 완성도 평가를 강조하고 있다.

본 연구에서는 이러한 기존 연구들을 바탕으로, 자연어 처리와 소스코드 분석을 결합한 새로운 완성도 평가 방법론을 제안하고자 한다. 완성도 평가는 기능 구현율, 코드 커버리지, 요구사항 구현 상태 등을 기준으로 수행된다.

3. 제안된 완성도 평가 프레임워크

제안된 프레임워크는 평가자 개입을 최소화하

면서, 요구사항의 충족 여부를 정량화할 수 있으며, 실시간 대시보드를 통해 객관적 근거를 제공함으로써 감정 효율성과 신뢰도를 동시에 확보하는 것이 목표이다. 본 프레임워크는 다음과 같은 파이프라인을 갖는다. 사용자 요구사항이 입력되면, NLP 기반 요구사항을 추출하고, 소스코드 분석 및 요구사항 매핑을 수행한다. 완성도 평가 기준을 설정하고 나면, 소프트웨어 완성도 평가가 이루어진다. 제안된 프레임워크를 이용한 자연어 요구사항 기반 소프트웨어 완성도 평가 절차는 다음 그림 1과 같다.

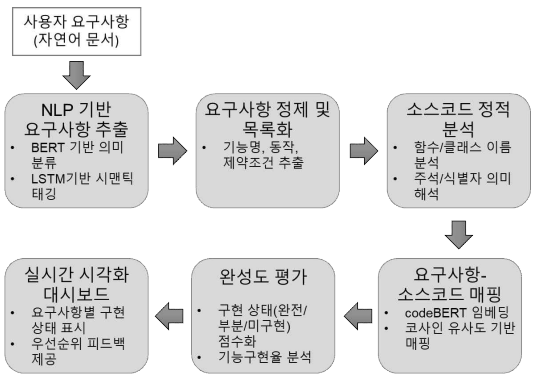

그림 1. 자연어 요구사항 기반 소프트웨어 완성도 평가 절차

Fig. 1. Software completeness evaluation procedure based on NLP

3.1 NLP 기반 요구사항 추출

자연어로 작성된 요구사항 문서에서 기능적 요구사항을 자동으로 추출하는 작업은 NLP 기법을 통해 이루어진다. 첫 번째 단계로, 요구사항 문서에서 각 문장을 분리하고, 기능적 요구사항을 식별한다. 여기에는 규칙 기반 키워드 패턴 매칭, Transformer 기반 문맥 분석 모델 (BERT fine-tuning), 시맨틱 분류기 (Text Classification using LSTM)을 사용한다. 문장 단위 분리는 파이썬 NLTK로 구현하였고, 한 문장에 여러 기능이 포함된 경우에도 이를 분할하여 다중 요구사항으로 추출이 가능하다.

요구사항 추출 과정은 자연어 요구사항 문장이 입력되면, BERT 토큰 임베딩 이후, Dense Layer와 Softmax Layer를 거쳐 기능/비기능/무관문장으로 분류하게 된다. 다음 표 1은 기능문장과 비기능문장의 사례이다.

표 1. 기능/비기능 문장의 사례

Table 1. An example of functional and non-functional requirements

기능문장	Members should be able to view their orders.
비기능문장	Response speed should be less than 2 seconds.

기능 내부 요소 식별을 위해 LSTM 기반 시퀀스 분류 모델을 사용한다. 전처리된 요구사항을 주체(Actor)-행위(Action)-대상(Object)을 나타내도록 (주체, 행위, 대상) 튜플로 목록화 한다. 이 구조는 이후 코드 매핑 시 임베딩 입력으로 사용하게 된다. 예를 들면, “Members should be able to view their orders.” 문장은 (Member, view, order)로 정의된다. 이를 통해 개발자는 요구사항 목록을 쉽게 이해할 수 있으며, 추후 소스코드와 매핑하는 작업이 용이해진다.

3.2 소스코드 분석 및 요구사항 매핑

소스코드 분석은 정적 분석 도구를 사용하여 각 코드의 기능을 파악하는 단계이다. 코드 내의 함수, 클래스, 메소드를 분석하여 해당 기능이 무엇을 수행하는지를 이해한다. 또한, 코드의 주석, 메소드 이름, 변수명 등을 기반으로 코드의 의도를 해석한다. 이전 단계에서 추출한 기능 내부

요소 목록을 소스코드의 메소드/함수와 매핑한다. 이를 위해 코드 유사도 분석 도구를 사용하여 요구사항과 소스코드 간의 연관성을 자동으로 파악한다. 매핑 된 요구사항과 코드를 비교하여 요구사항이 구현되었는지, 또는 부분적으로 구현되었는지를 확인한다. 기본 모델로 CodeBERT[14]를 사용하였고, 요구사항 문장과 코드 블록을 각각 임베딩하고, Siamese Network 구조로 두 벡터 간 코사인 유사도 계산한다. 유사도 임계치(Threshold)는 0.85를 사용했고, 손실 함수는 Contrastive Loss를 사용했다. CodeBERT 기반 요구사항-소스코드 매핑 과정은 다음 그림 2와 같다.

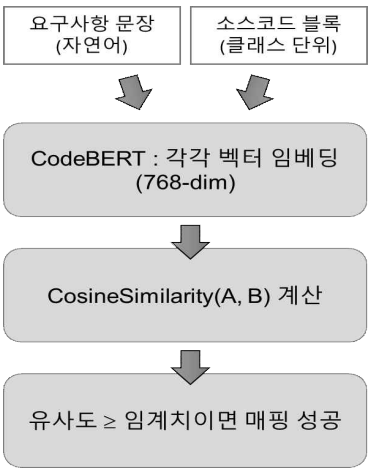

그림 2. 요구사항-코드 매핑과정
Fig. 2. Requirement-to-Code matching

3.3 소프트웨어 완성도 평가

기능 구현율은 요구사항이 얼마나 구현되었는지를 평가하는 가장 기본적인 지표이다. 전체 요구사항 중 실제로 구현된 요구사항의 비율을 계산하여 소프트웨어의 완성도를 평가한다. 예를 들어, 전체 요구사항 20개 중 16개가 구현되었다면, 기능 구현율은 80%가 된다.

Cpl_score = (Im_score / Tot_score) * 100

위 식에서 Cpl_score는 기능구현율을 의미하며, Im_score는 구현된 항목의 개수, Tot_score는 전체 요구사항의 개수를 나타낸다.

본 논문에서는 구현된 비율이 아닌, 각 요구사항의 구현 상태에 따라 가중치를 부여하여 완성도를 평가한다. 각 요구사항에 대해 완전 구현, 부분 구현, 미구현 상태를 점수화하여 소프트웨어의 전체 완성도 점수를 산출한다. 각 요구사항은 매핑된 요소의 수와 유사도에 따라 완전구현, 부분구현, 최소구현으로 구현상태가 결정되며, 완전 구현은 5점, 부분 구현은 3점, 최소 구현은 1점으로 할당한다. 이 점수를 통해 소프트웨어의 전반적인 품질을 측정할 수 있다.

overall_Cpl = (Tot_score / Max_score) * 100

위 식에서 overall_Cpl은 전체 완성도 점수를 의미하며, Tot_score는 각 요구사항에 대해 구현 상태에 따라 계산된 점수이며, Max_score는 기능목록이 모두 완전구현된 경우 가질 수 있는 최대 값이다.

완성도 평가 결과를 분석하고 나면, 개발자에게 우선순위에 따른 수정 사항 피드백을 제공할 수 있다. 미구현된 요구사항은 수정할 우선순위를 설정하고, 부분적으로 구현된 요구사항은 개선을 위해 필요한 작업을 제시한다.

4. 사례 연구 및 검증

본 장에서는 제안된 방법론을 실제 사례에 적용하여 그 유용성을 검증하고, 이 방법론이 실제 소프트웨어 개발 환경에서 어떻게 효과적으로 활용될 수 있는지를 분석한다. 사례 연구는 실용적인 관점에서 방법론의 정확성과 효율성에 초점을 맞추어 진행하였다. 제안된 프레임워크의 프로토

타입은 파이썬과 PyTorch, Transformers, scikit-learn, streamlit, HuggingFace Datasets 등을 이용하여 구현되었다. 다음 그림 3과 그림 4는 각각 구현된 프로토타입의 초기 화면과 실시간으로 제공되는 시각화 대시보드의 화면이다.

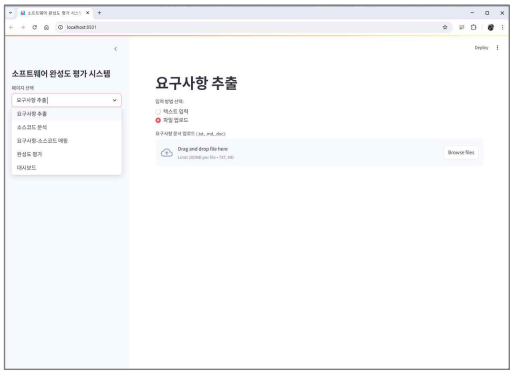

그림 3. 프로토타입의 초기 화면
Fig. 3. The initial screen of the prototype

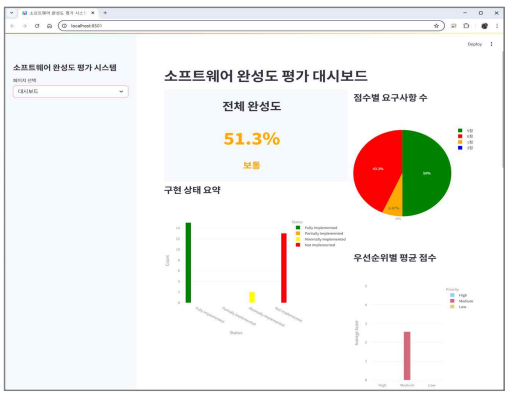

그림 4. 시각화 대시보드
Fig. 4. Visualization dashboard

4.1 사례 연구

본 연구에서는 전자상거래 웹 애플리케이션의 소프트웨어 개발 프로젝트를 선택하여 제안된 방법론을 적용하였다. 이 프로젝트는 고객 주문 처리, 결제, 상품 검색 기능 등 총 30개의 기능적 요구사항을 포함하고 있다. 본 연구에서는 자연어 처리 기술을 사용하여 요구사항 문서에서 각 기능적 요구사항을 추출하고, 이를 목록화하였다. 자연어 처리 기법을 통해 요구사항을 추출하는 과정에서 문장 분석, 형태소 분석, 구문 분석을 진행하였다. 예를 들어, "Users should be able to add products to their shopping cart."는 요구사항은 "user"와 "cart"를 핵심 개념으로 추출하고, 이와 관련된 행동을 동사로 파악하여 구현할 기능을 정의했다. 추출된 요구사항 목록은 소스코드와 매핑을 위한 기초 자료로 사용된다. 다음 그림 5는 요구사항 파일이 업로드 되어 요구사항이 추출된 결과를 보여주고 있다. 요구사항 파일은 30개 기능적 요구사항이 입력된 텍스트 파일이다.

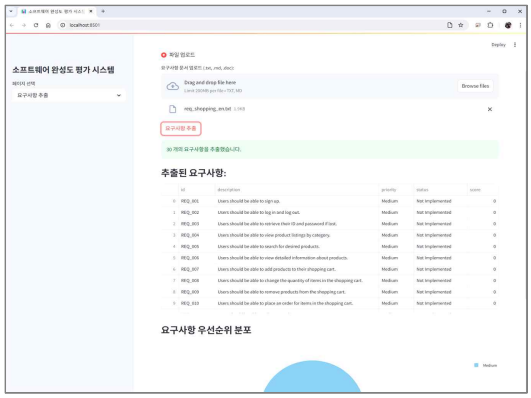

그림 5. 요구사항 추출 결과
Fig. 5. Extracted requirements display

요구사항이 추출되고 나면, 전자상거래 애플리케이션의 소스코드를 정적 분석하여, 각 기능적 요구사항이 실제로 구현된 부분을 식별한다. 예를 들어, "search for desired products" 요구사항은 검색 API와 검색 결과 페이지를 연결하는 코드 블록과 매핑된다. 또한, 결제 시스템 관련 요구사항은 결제 API와 관련된 메소드들을 매핑하

여 구현된 상태를 확인했다. 요구사항 "Users should be able to remove products from the shopping cart."는 소스코드 내에서 장바구니 관리 클래스의 removeItem() 메소드와 매핑되었으며, 이 메소드가 요구사항을 구현한 코드로 확인되었다. 소스코드를 분석한 결과는 그림 6과 같이 표와 그래프의 형태로 제공된다. 각 기능에 대해 매핑된 코드의 함수명/메소드명을 나열하고, 코드에 나타난 함수와 클래스, 메소드 개수 등에 대한 통계를 제공한다.

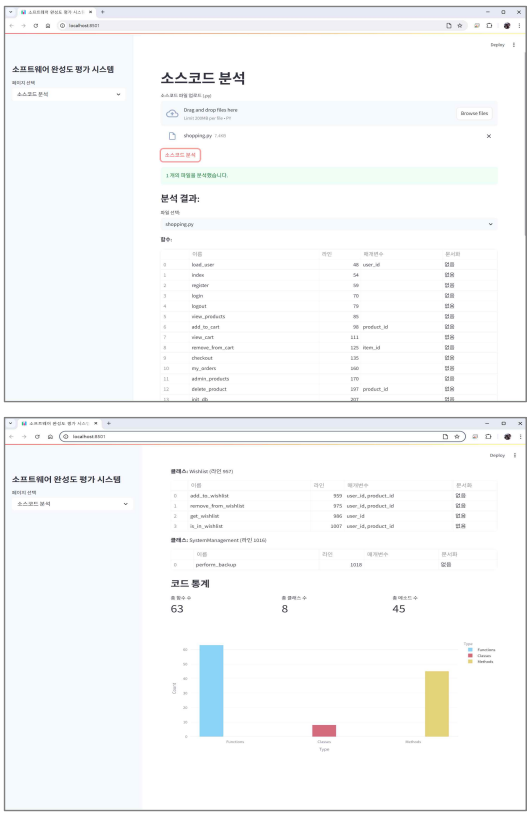

그림 6. 소스코드 분석 결과
Fig. 6. Source code analysis results

각 요구사항의 구현 상태를 기준으로 점수를 부여하여 소프트웨어 완성도를 평가하였다. 예를 들어, "장바구니에서 상품을 삭제할 수 있어야 한다."는 요구사항은 완전구현되었으므로 5점을 부여하였고, "결제 시 여러 결제 방법을 제공해야 한다."는 요구사항은 부분적으로 구현되었으므로 1점을 부여했다. 가중치는 적용 대상 시스템에 따라 동적으로 변경이 가능하다.

전체 30개의 요구사항 중 15개가 완전구현되었고, 2개는 최소구현, 13개는 미구현 상태였다. 이들 상태에 따라 각 요구사항 목록에 대해 score가 5점, 3점, 1점이 할당된다. 전체 30개 요구사항에 대해 구현 상태 별 비율은 다음 표 2와 같이 평가되었다.

표 2. 구현 상태 별 비율
Table 2. Proportion by implementation status

구현 상태	개수	비율(%)
완전구현	15	50
최소구현	2	6.67
미구현	13	43.3

따라서 전체 기능 구현율은 (77 / 150) * 100 = 51.3%로 평가되었다. 아래 그림 7은 완성도 평가 결과를 나타낸다. 평가 결과를 나타내는 게이지 차트는 전체 완성도를 표현하고 있다.

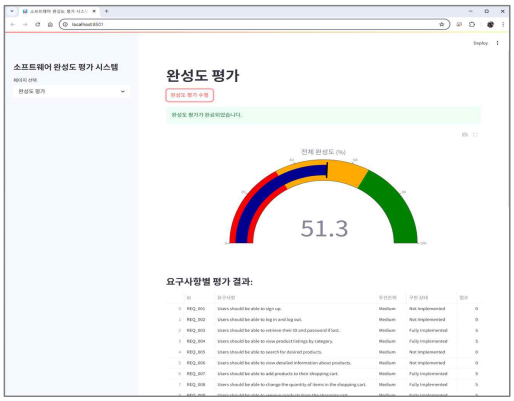

그림 7. 완성도 평가 결과 요약 화면
Fig. 7. Evaluation score overview display

요구사항 누락 사례를 분석한 결과, 70%는 '명확하지 않은 자연어 표현'에서 기인했고, 나머지는 유사한 코드가 여러 함수에 분산되어 있는 경우였다. 이는 향후 요구사항 명세 품질 향상 도구와의 연계를 통해 보완할 수 있을 것이다.

4.2 완성도 평가 방법의 효과성 검증

본 연구의 목적은 제안된 프레임워크를 활용하여 요구사항을 자동으로 매핑하고, 평가 결과를 시각적으로 제공하는 대시보드를 구축함으로써, 소프트웨어 완성도 평가를 수행하는 감정인의 작업 효율성 및 평가 결과의 일관성을 제고하는 데 있다. 제안된 방법의 유용성을 검증하기 위해, 소프트웨어 감정 수행 경험이 있는 10명의 전문가를 대상으로 프로토타입에 대한 설문 조사를 실시하였다. 그 결과, 표 3과 같이 소프트웨어 감정 시간 및 일관성 변화를 살펴볼 수 있었다.

표 3. 감정사항의 결과
Table 3. results of evaluation

항목	기존 수작업 감정	제안 프레임워크 활용	변화율
프로젝트 당 평균 평가 소요시간	6시간 30분	4시간 5분	-37%
감정인 간 Kappa 일치도	0.82	0.93	+13%

평가자들의 피드백에는 "이전보다 비교적 근거가 명확하며, 수작업 오류를 어느 정도 줄일 수 있다"는 의견이 있었다. 한계점으로는 자연어 요구사항의 모호한 표현에 따른 불명확성과, 매핑되지 않는 비기능 요구에 대한 정량화 부족이 지적되었다.

감정인 외에도 실제 개발자 및 QA 담당자 10명을 대상으로 System Usability Scale 기반 평가를 수행한 결과, 100점 기준 평균 86.5를 기록하며 높은 실용성을 입증하였다. 특히 실시간 대시보드 기능이 오류 수정 우선순위 설정에 유용하다는 피드백이 있었다. 이상으로, 제안한 프레임워크가 기존 수작업 기반의 소프트웨어 완성도 평가를 보조하며, 정량적 지표와 자동화 기술을 통해 평가의 신뢰성과 효율성을 크게 향상시킬 수 있음을 확인하였다. 요구사항 추출과 소스코드 매핑을 자동화하여 소프트웨어 품질 관리에 큰 도움이 되지만, 일부 복잡한 요구사항이나 기술적인 난이도가 높은 요구사항의 경우, 자동화 도구의 정확도가 떨어질 수 있었다. 또한 한글로 작성된 요구사항의 경우, 소스코드와의 자동 매핑의 정확성이 크게 떨어지는 문제는 향후 개선된 모델을 통해 보완할 필요가 있다.

5. 결론 및 향후 연구과제

본 연구에서는 자연어 요구사항과 소스코드 검증을 통해 소프트웨어의 완성도를 평가하는 방법론을 제시하였다. 이 방법론은 요구사항 추출, 소스코드 분석, 완성도 평가를 자동화하여 실시간으로 소프트웨어 품질을 모니터링하고 피드백을 제공한다. 또한 실제 사례에 대한 적용을 통해 제안된 방법론은 소프트웨어 완성도 평가에 있어 유용한 도구임을 입증하였다. 자연어 요구사항 추출과 소스코드 매핑을 자동화하여, 개발자는 실시간으로 요구사항 구현 상태를 확인하고 피드백을 받아 개선할 수 있었다. 또한, 완성도 평가를 통해 소프트웨어 품질을 관리할 수 있었으며, 이 방법론은 향후 다양한 소프트웨어 개발 환경에서 적용될 수 있는 가능성을 가지고 있다.

향후 연구에서는 본 프레임워크의 요구사항-소스코드 매핑 정확도 향상을 핵심 과제로 삼고자 한다. 현재 사용된 CodeBERT 기반 임베딩은

일반적인 기능 구현 판단에는 유효하나, 한글처리에 한계성이 있으므로 다음과 같은 고도화 방안을 통해 더욱 정밀한 추적성과 평가 신뢰도를 확보할 수 있을 것으로 본다.

(1) 다중 소스 기반 맥락 분석: 자연어 요구사항만이 아닌 UI 문서, API 명세서, 테스트 케이스 등 다양한 문서와의 교차 정보를 활용하여, 요구사항의 의미적 해석 범위를 넓히는 전략을 도입할 예정이다.

(2) 고성능 LLM 기반 코드 이해 모델 적용: GPT-4, CodeWhisperer, StarCoder 등 LLM의 파인튜닝을 통해 문맥 보존 능력과 특히 한글과 소스코드의 정밀 매핑 능력을 향상 시킬 계획이다. 특히, 부분구현과 최소구현을 정확히 구분하기 위해, 코드 시맨틱을 판단할 수 있는 기법의 적용을 고려하고 있다.

(3) 비정형 요구사항 자동 분류 강화: 모호한 표현을 포함한 비정형 문장에 대해 의미 명확화를 수행하는 전처리 모듈의 강화가 요구된다. 이와 관련하여, 요구사항 품질 분류와 모호성 탐징 모델을 병렬적으로 연결하는 구조를 적용할 예정이다.

이러한 정밀도 향상 전략은 추후 다양한 도메인(금융, 헬스케어, 공공)에서의 적용 시, 신뢰성과 법적 근거성을 높이는 데 중요한 기반이 될 수 있을 것이다. 또한 전자상거래 프로젝트에 한정된 사례를 제시하였으나, 향후 금융, 헬스케어, 교육 등 다양한 산업 도메인의 복잡한 요구사항을 포함한 환경에 이 프레임워크를 적용하여 일반화를 검증할 계획이다.

참 고 문 헌

[1] I. Dikmen, G. Eken, H. Erol, M. T. Birgonul, “Automated construction contract analysis for risk and responsibility assessment using natural language processing and machine learning”, Computers in Industry, Vol. 166, 104251, April 2025. DOI: https://doi.org/10.1016/j.compind.2025.104251

[2] “Natural Language Processing in Contract Analysis: Enhancing Due Diligence Efficiency”, April 2024. https://createprogress.ai/natural-language-processing-in-contract-analysis-enhancing-due-diligence-efficiency/

[3] S. Ghosh, D. Elenius, W. Li, P. Lincoln, N. Shankar, W. Steiner, “ARSENAL: Automatic Requirements Specification Extraction from Natural Language”, Lecture Notes in Computer Science, vol 9690. Springer, Cham. DOI: https://doi.org/10.1007/978-3-319-40648-0_4

[4] A. Nayak, H. P. Timmapathini, V. Murali, K. Ponnalagu, V. G. Venkoparao, A. Post, “Req2Spec: Transforming Software Requirements into Formal Specifications Using Natural Language Processing”, In Proceedings of the International Conference on Requirements Engineering: Foundation for Software Quality(REFSQ 2022), pp.87-95, Birmingham, UK, March 21 - 24, 2022. DOI: https://doi.org/10.1007/978-3-030-98464-9_8

[5] A. Veizaga, S. Y. Shin and L. C. Briand, “Automated Smell Detection and Recommendation in Natural Language Requirements”, IEEE Transactions on Software Engineering, vol. 50, no. 4, pp. 695-720, April 2024. DOI: https://doi.org/10.1109/TSE.2024.3361033.

[6] Q. Motger, X. Franch, “Automated Requirements Relations Extraction”, In: Ferrari, A., Ginde, G. (eds) Handbook on Natural Language Processing for Requirements Engineering, pp.177-206, Springer, Cham. 2025. DOI: https://doi.org/10.1007/978-3-031-73143-3_7

[7] J. Frattini, M. Junker, M. Unterkalmsteiner, D. Mendez, “Automatic extraction of cause-effect-relations from requirements artifacts”, In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20), pp.561 - 572, 2020.
DOI: https://doi.org/10.1145/3324884.341654
[8] S-C Necula, F. Dumitriu, V. Greavu-Şerban, “A Systematic Literature Review on Using Natural Language Processing in Software Requirements Engineering”, Electronics, vol. 13, no. 11, pp.2055, 2024. DOI: https://doi.org/10.3390/electronics13112055
[9] Ra'Fat Al-Msie'deen, “Requirements Traceability: Recovering and Visualizing Traceability Links Between Requirements and Source Code of Object-oriented Software Systems”, International Journal of Computing and Digital Systems, Vol. 14, No.1, pp.279-295, Jul. 2023. DOI: https://doi.org/10.48550/arXiv.2307.05188
[10] X. Chen, J. Hosking, J. Grundy, et al. “DCTracVis: a system retrieving and visualizing traceability links between source code and documentation”, International Journal of Automated Software Engineering, vol. 25, pp.703 - 741, 2018. DOI: https://doi.org/10.1007/s10515-018-0243-8
[11] P. Dai, L. Yang, Y. Wang, D. Jin, Y. Gong, “Constructing Traceability Links between Software Requirements and Source Code Based on Neural Networks”, Mathematics, vol. 11, no. 2, pp.315, 2023. DOI: https://doi.org/10.3390/math11020315
[12] T. O. Hovorushchenko, “Methodology of Evaluating the Sufficiency of Information for Software Quality Assessment According to ISO 25010”, Journal of Infomation and Organizational Sciences, vol. 42, no. 1, Jun. 2018. DOI: https://doi.org/10.31341/jios.42.1.4
[13] D. Kim, “Software Completeness Evaluation based on ISO/IEC9241.10”, Journal of Software Assessment and Valuation, vol. 15, no. 2, pp.9-16, 2019. DOI: https://doi.org/10.29056/jsav.2019.12.02
[14] Z. Feng, et al., “CodeBERT: A Pre-Trained Model for Programming and Natural Languages”, The Association for Computational Linguistics, EMNLP, pp. 1536-1547, 2020. DOI: https://doi.org/10.18653/v1/2020.findings-emnlp.139

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