소프트웨어 수명 주기의 각 단계에 생성형 AI를 접목한 소프트웨어 개발 방안

진예진[1], 김기두[2], 유동영[3], 김영철[4]
[1,3,4]홍익대학교 소프트웨어융합학과
[2]한국정보통신기술협회
[1]yejin_jin@g.hongik.ac.kr, [2]kdkim@tta.or.kr, [3]ydy@hongik.ac.kr, [4]bob@hongik.ac.kr

Adopting Generative AI in Each Phase of Software Life Cycle for Software Development Approach

Ye-Jin Jin[1], Ki-Du Kim[2], Dong-Young Yoo[3], R. Young Chul Kim[4]
[1,3,4]Dept. of Software and Communications Engineering, Hongik University
[2]Telecommunication Technology Association

요 약

최근 대규모 언어 모델의 발전으로 대부분의 소프트웨어 개발자는 AI 코드 보조 도구를 활용하는 추세이다. 그러나, AI 기반 코드는 품질 보장 및 검증이 필요하다. 이 문제를 소프트웨어 공학적 시도로, 단계적 AI 적용으로 증명하고자 한다. 개발 프로세스 내에서 요구사항 단계에 AI를 적용하여 분석하고, 설계를 추출하고, 코드를 자동 생성하고자 한다. 즉, 자연어 기반 코드 자동화 메커니즘을 제안한다. 이를 위해 1) 요구공학 분야에 LLM을 접목하여 체계적인 분석, 2) UML 설계 추출, 3) 모델 변형 기법을 통해 코드 생성 자동화를 수행한다. 본 연구는 전통적인 방법의 확장으로 자연어 요구사항 중복성 제거 및 설계 메타모델을 통합한다. 이를 통해 코드와 테스트 케이스의 중복을 줄일 수 있다. 우리는 AI 기법을 소프트웨어공학의 단계적 절차에 접목함으로써 인간 중심의 코드와 AI 기반 코드 비교 검증을 기대한다.

1. 서론

현재 대부분의 개발자는 AI 도구로 초기 코드 베이스 생성 및 코드 완성, 오류 수정을 수행한다[1]. 그러나, 이들은 개발의 시간적 비용을 줄이는 것에만 집중한다. AI 코드는 불확실한 내부 알고리즘을 통해 생성된다. 따라서, 절차적 코드 발생 과정 검증 및 코드 품질 보장을 할 수 없다. 기존 연구에서는 소프트웨어 개발 생명 주기를 기반으로, 생성형 AI를 부분적으로 적용하는 메커니즘을 제안했다[2]. 이 방법은 1) 촘스키의 구문 구조 분석 이론과 필모어의 의미론적 분석 이론에 따라 자연어 요구사항을 분석한다. 2) 요구사항 분석 결과를 UML 요소와 매핑하여 설계를 생성하고, 3) 메타모델링을 통해 설계로부터 코드를 발생한다.

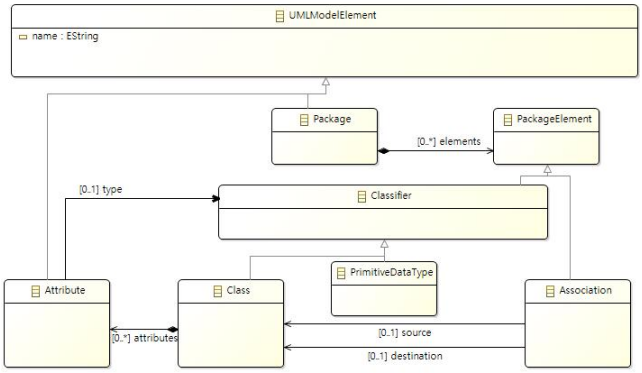
본 연구는 기존 연구의 메커니즘을 개선한다. 설계 생성 과정에 중복성 제거하고, 코드 발생을 위한 설계 메타모델을 정의한다. 이를 통해 중복 코드를 제거하고자 한다. 또한, 테스트 케이스의 중복성도 고려될 수 있다.

2장에서는 UML 메타모델링 기반 코드 발생 관련 선행 연구를 설명하고, 3장에서는 개선된 메커니즘을 제안한다. 4장에서는 사례 연구를 언급하고, 마지막으로 결론을 언급한다.

2. 기존 UML 기반 코드 발생 방법

기존 연구는 모델 기반 코드 생성을 보다 효율적으로 수행하기 위해 ExUML을 제안하였다[3]. ExUML은 UML 다이어그램으로부터 실행이 가능한 소프트웨어를 생성하는 추상적 모델링 언어이다. 이는 코드 생성을 지원하는 모델 중심 소프트웨어 개발 접근법이다. 그림 1은 연구 프로세스의 핵심을 보여준다. 문제는 UML 다이어그램을 단순한 정적 자료구조로 매핑하고 저장하여, 모델의 의미나 동작을 해석하기 어렵다.

다른 연구에서는 모델 변형 자동화를 활용하였으나, 대부분 하나의 UML 모델을 사용하였다[4]. 이는 여러 설계 모델을 통합하지 못하고, 여러 개의 메타모델과 그에 따른 모델 변환 규칙이 필요하다.

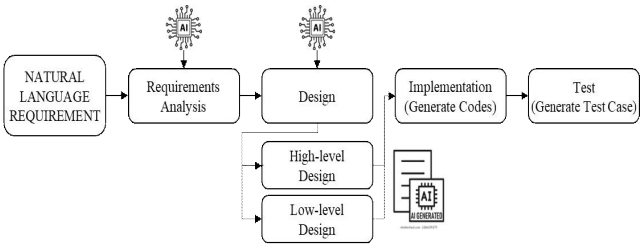


(그림 1) 클래스 다이어그램 메타 모델

따라서 본 연구에서는 클래스 다이어그램, 순차 다이어그램, 상태 다이어그램의 설계 모델을 통합적으로 표현할 수 있는 설계 메타모델을 새롭게 정의한다. 하나의 메타모델 변환 규칙을 사용하여 모델 간의 의미적 연결을 유지한다.

3. AI와 소프트웨어공학 접목

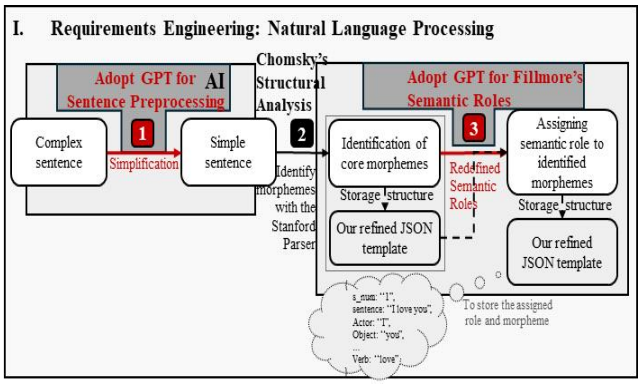
기존 소프트웨어공학에서는 요구사항 → 분석 → 설계 → 구현 → 테스팅 단계를 통해 수행한다. 제안한 방법은 소프트웨어 개발 프로세스 내의 단계별 AI 기법 접목이다. 제안한 절차는 다음과 같다.



(그림 2) 단계별 AI를 접목한 소프트웨어 개발 프로세스

3.1. 생성형 AI를 적용한 요구공학

기존 연구에서 LLM을 요구공학에 접목하였다[2]. 개선된 방법으로 LLM 기반 자연어 요구사항 분석 단계는 그림 3과 같다.



(그림 3) LLM을 활용한 자연어 요구사항 분석

자연어 요구사항의 분석은 3가지의 단계를 거친다. 그림 3의 1번과 3번 단계에서는 기존에 규칙 기반으로 분석하기 어렵다. 따라서, LLM 프롬프팅 기술을 통해 GPT를 활용하여 자동화한다.

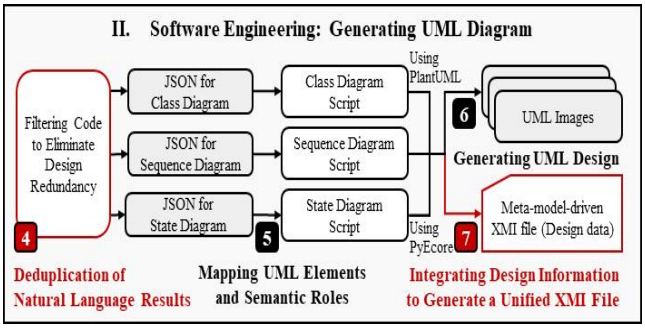
Step 1) 요구사항 문장 전처리: 복잡한 문장은 분석 결과 또한 복잡하여 문장 정보를 처리하기 어렵다. 따라서 여러 개의 주어와 동사가 있는 복문과 중문을 단문으로 변환한다.

Step 2) 문장 구조 및 형태소 분석: 문장의 구조와 의미 해석을 위해 촘스키의 구문 구조 분석 이론을 사용한다[5]. NLTK(Natural Language Toolkit) 라이브러리를 통해 문장을 가장 작은 단위인 형태소로 나누고 품사 태그를 통해 명사, 동사, 형용사를 구분한다.

Step 3) 문장 의미 분석: 분석된 형태소의 품사에 따라 문장의 의미를 분석한다. 필모어의 의미역 이론을 사용하여 동사를 기준으로 각 명사의 관계를 분석한다[6]. 재정의한 필모어의 역할론을 바탕으로 명사에 역할을 부여한다.

3.2. 요구사항 데이터 기반 UML 설계 생성

자연어 요구사항으로부터 UML 설계를 추출한다. 클래스, 순차, 상태 다이어그램을 생성한다. 그림 4는 자연어 요구사항 분석 데이터를 바탕으로 3가지의 UML 설계 생성 과정을 보여준다.



(그림 4) UML 설계 생성 과정

그림 4의 4번과 7번 단계는 기존 연구에서 확장되었다. 기존 연구[2]에서는 자연어 요구사항의 중복을 고려하지 않았다. 따라서 4번 단계를 추가하여 중복성을 제거한다. 기존 연구의 설계 메타모델은 선행 연구에서 사용된 구조를 따르지만, 통합 메타 모델로 개선하였다.

Step 4) 설계 중복 제거를 위한 필터링: 요구사항의 중복성은 결국 설계 및 코드의 중복으로 이어진다. 중복 코드는 코드의 길이가 늘어나고, 복잡성을 증가시킨다. 따라서, 개발 초기에 중복성을 없애

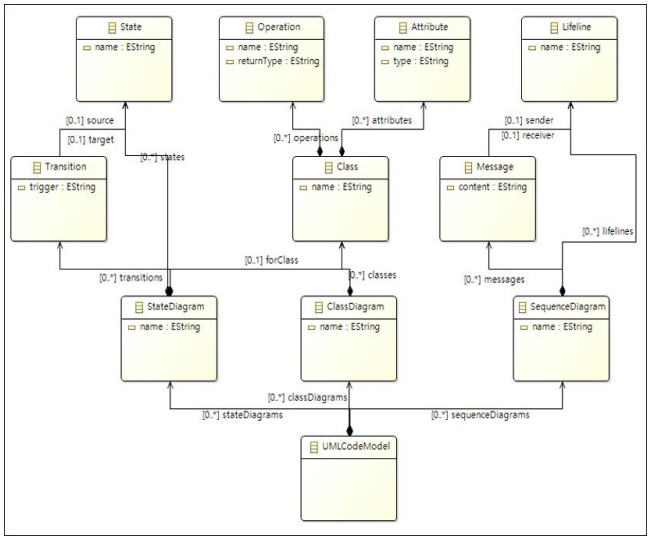
는 것은 매우 중요하다. 자연어 요구사항의 분석 결과에서 중복된 명사 및 동사가 여러 개일 때 하나로 처리하는 과정을 거친다. 중복을 제거한 데이터는 다이어그램 생성에 필요한 JSON 파일로 저장한다.

Step 5) UML 설계 요소와 요구사항 분석 결과 매핑: 생성될 UML 설계는 3가지이다. 각 다이어그램에 필요한 요소를 정의하고, 이를 규칙 기반으로 요구사항 결과와 매핑한다.

Step 6) UML 이미지 생성: 다이어그램 생성을 위해 PlantUML을 사용한다. 매핑된 요구사항 데이터를 바탕으로 다이어그램 생성 스크립트를 정의하고, 이미지로 생성된다.

Step 7) 코드 생성을 위한 설계 데이터 저장: 본 연구에서 재정의된 UML 설계 메타모델을 바탕으로 요구사항 결과와 매핑된 다이어그램 요소가 저장된다. 기존 연구에서는 각 다이어그램의 메타모델이 존재하여 3개의 설계 메타모델에 대해 데이터 파일 생성, 데이터 입력, 그리고 데이터 변환하는 과정이 각각 필요하다.

우리는 이를 하나의 데이터 파일로 병합하여 정보를 저장한다. 하나의 설계 메타모델을 설계하여 이러한 과정을 단순화한다. 그림 5는 통합한 UML 메타모델이다.

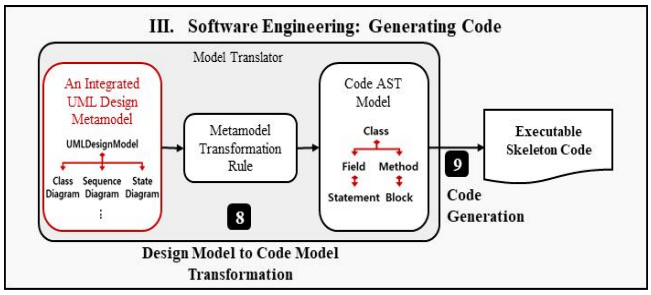


(그림 5) 재정의된 UML 설계 메타모델

설계 메타모델의 최상위 루트노드는 'UMLDesignModel'이다. 각각의 UML 설계의 상위 노드들이 상속될 수 있도록 한다. 그림 5의 UML 메타모델 구조를 기반으로 XMI 파일을 구성 및 저장한다.

3.3. 메타모델링 기법을 통한 스켈레톤 코드 생성

다이어그램 설계가 생성되면, 이를 바탕으로 실행이 가능한 스켈레톤 코드를 생성한다. 그림 6은 메타모델링 기법을 사용하여 설계데이터로부터 코드를 생성하는 과정을 보여준다.



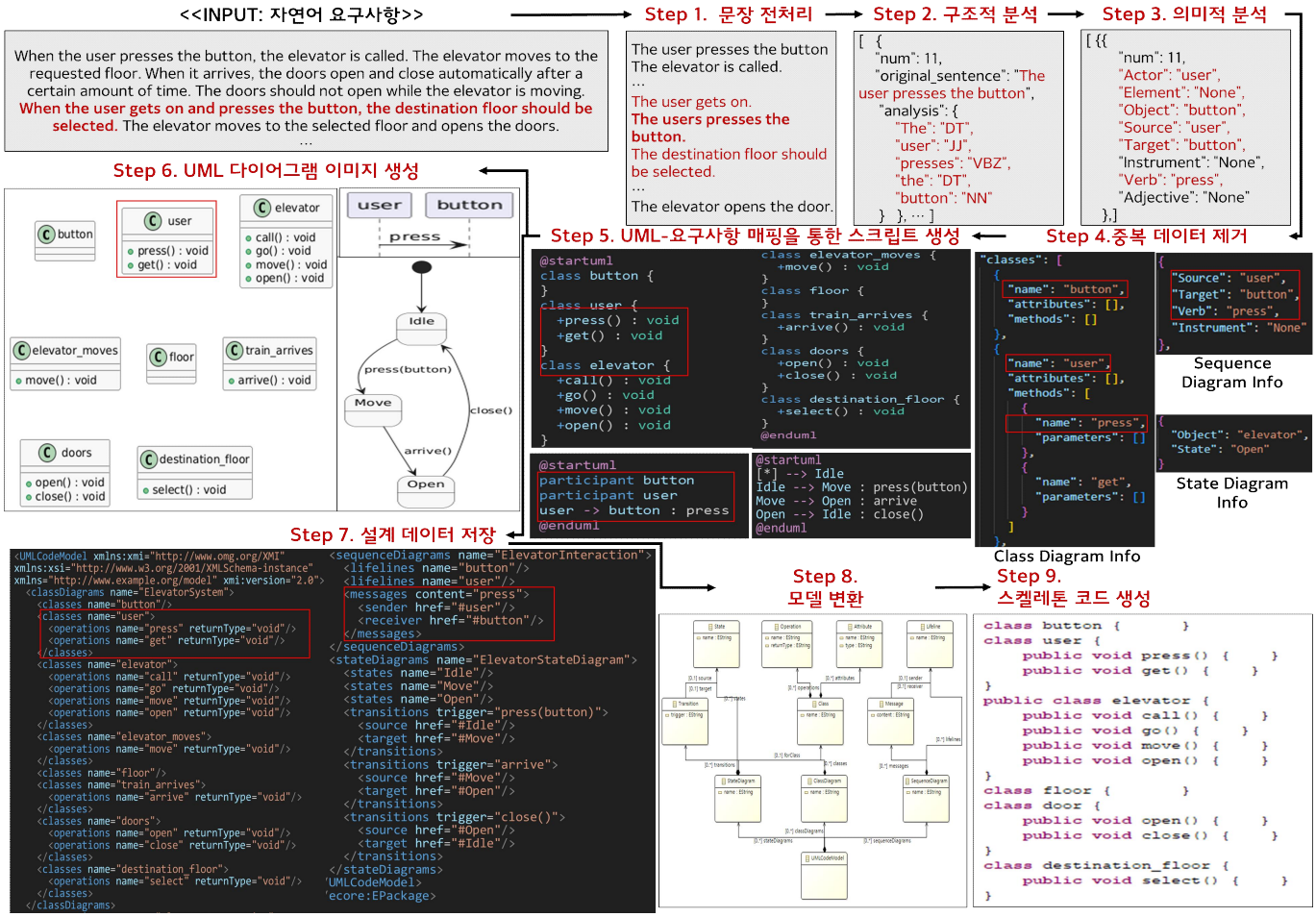
(그림 6) 메타모델링을 통한 코드 생성

Step 8) 메타모델링 기법을 활용한 모델 변환: 본 연구에서는 Java 언어를 기반으로 코드를 생성한다. Java의 문법 구조는 트리 형태로 나타나는 AST(Abstract Syntax Tree) 구조를 가진다. 이를 기반으로 Code AST 메타모델을 생성한다. UML 설계 메타모델과 코드 메타모델의 데이터는 규칙 기반으로 데이터 변환을 수행한다.

Step 9) 실행 가능한 스켈레톤 코드 생성: 모델 변환을 통해 설계 메타모델의 데이터를 코드 메타모델의 데이터로 저장한다. 우리는 Java Eclipse 환경에서 연구를 진행한다. Eclipse JDT(Java Development Tools) 내장 자바 파서와 AST를 활용하여 실행 가능한 스켈레톤 코드를 생성한다.

4. 사례 연구

제안한 연구 프로세스를 간단한 엘리베이터 시스템으로 설명한다. 그림 7은 요구사항 한 문장을 중점적으로 설명한다. 절차에 맞게 문장을 전처리, 구조 분석, 의미 분석을 진행하고, 각 단계에 따라 json 파일을 생성한다. 이후, 의미 분석 파일에서 동일 역할을 수행하는 데이터를 하나로 처리하여 중복 데이터를 제거한다. 이를 바탕으로 새로운 다이어그램 데이터 파일을 만든다. 데이터 파일로부터 Plant UML 스크립트를 작성하고, 다이어그램을 생성한다. 또한, 사전 정의한 메타 모델 구조에 따라 xmi 파일을 구성하고, 다이어그램 데이터 파일의 요소를 저장한다. 사전 정의된 메타 모델은 Ecore 모델로 저장되어있고, 우리가 생성한 xmi 파일을 바탕으로 모델 변환을 수행하여 코드를 생성한다. 따라서 우리는 자연어 요구사항으로부터 설계 생성 코드 생성까지 일관성 있는 개발이 가능하다.



(그림 7) 자연어 요구사항 분석으로부터 설계 및 코드 생성 사례

5. 결론

본 연구는 AI와 소프트웨어공학의 접목 방안을 고려한다. 즉, 소프트웨어 개발 생명 주기 (요구사항 → 분석 → 설계 → 코드 생성)에 따라 요구사항 분석 단계별로 인공지능을 적용했다. 또한, 요구사항의 중복성을 제거하고, 자동 모델 변형을 위해 설계 메타모델을 재정의하였다.

향후 연구는 모든 개발 단계에서 AI를 접목하고자 한다. 그 결과로, 1) 인간 기반의 코드와 2) 부분적 AI 적용된 코드, 그리고 3) 완전히 AI 기반 코드를 체계적으로 비교 분석으로, 세 가지 방법에서 고품질의 개발 방식을 식별 및 검증하고자 한다.

Acknowledgement

본 연구는 2025년도 문화체육 관광부의 재원으로 한국콘텐츠진흥원(과제명: 인공지능 기반 대화형 멀티모달 인터렉티브 스토리텔링 3D장면 저작 기술 개발, 과제번호: RS-2023-00227917, 기여율:100%) 지원과 한국연구재단의 4단계 두뇌한국21사업(과제명: 초분산 자율 컴퓨팅 서비스 기술 연구팀, 과제번호: 202003520005)의 지원을 받아 수행된 연구임.

참고문헌

[1] Stack Overflow, AI | 2024 Stack Overflow Developer Survey, Retrieved April 17, 2025, from https://survey.stackoverflow.co/2024/ai.

[2] 진예진, 서채연, 공지훈, 김기두, 김영철, KCSE 2025, vol. 27, no. 1, pp. 289-292, 2025.

[3] Kim W.Y., Son H.S., Park Y.B., Park B.H.; Carlson C.R., R.Y.C Kim, Automatic MDA(Model Driven Architecture) Transformations for Heterogeneous Embedded Systems, SERP 2008, vol. 2, pp. 409-414, 2008.

[4] Son H.S., R.Y.C. Kim, Automatic transformation tools of UML design models from virtual prototypes of multi-jointed robots, Multimedia Tools and Applications, vol 77, no 4, pp. 5083-5106, 2018.

[5] Noam Chomsky, Syntactic structures, USA, Mouton de Gruyter, 2002.

[6] Charles J. Fillmore, “The Case for Case,” New York, HR&W, 1968.